Using Geographic Locations in Process Visualization

Smit, Bart

Award date:
2020

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Algorithms, Geometry, and Applications Cluster

Using Geographic Locations in Process Visualization

Master Thesis

Bart Smit

Supervisors:
dr. Wouter Meulemans
dr. ir. Roeland Scheepens

Committee:
dr. Wouter Meulemans
dr. ir. Roeland Scheepens
dr. Michael Burch

Eindhoven, September 2020

Abstract

In current process mining techniques, the geographic information of a process is not considered. This information could, however, add an extra dimension to the process visualization and help gain more insights. Therefore, we propose a framework which divides processes into four categories based on the cardinality relation between the activities and locations of the events in the event log corresponding to that process. We propose a clustering algorithm to remove the overlap between nodes in the visualization of the many-to-many category. The clustering algorithm chooses which nodes should be merged based on the position of that node in the region hierarchy. A quantitative analysis concludes that our algorithm performs worse on the metrics considering the change in node placement. However, our method performs better on stability in some cases. In addition, we propose an edge drawing algorithm for the visualization of the many-to-many category to remove the node-edge crossings. The edge drawing algorithm first reroutes the edges to remove node-edge crossings by using a tangent visibility graph. Afterwards, the algorithm displaces the edges around the circle to reduce the number of edge crossings around the circles. A quantitative analysis concludes that the other graph drawing aesthetics regarding edge drawing are not significantly impacted by our algorithm.

Preface

The results of my master thesis *Using Geographic Locations in Process Visualization*, performed at UiPath Process Mining are presented in this report. This thesis is the last part of my master Computer Science and Engineering at the Eindhoven University of Technology.

First, I want to thank Roeland Scheepens, my daily supervisor at UiPath Process Mining, for his extensive feedback, guidance, and for always willing to help or have a discussion throughout the project. I also want to thank Wouter Meulemans, my university supervisor, for the extensive feedback, discussions and advice throughout this project. I also want to thank Michael Burch for being part of the assessment committee. I also want to thank my colleagues at UiPath Process Mining for always willing to help and for the, sometimes needed, distraction.

Finally, I want to thank my family for their support and encouragement throughout my studies.

Contents

Chapter 1

Introduction

Almost every company has some form of process that needs to be executed. There are a lot of different types of processes. For example, a process for the production of a product or a process for paying an invoice. For a large company, these processes could be spread out over different departments or offices. In that case, it becomes difficult to determine how the process is exactly executed. Process mining is used to gain insights and visualize those processes. These insights could be used to determine bottlenecks in the process. Section 1.1 describes process mining in more detail.

For big companies, these processes can go through different offices. Those offices could be spread out over multiple countries. With current process mining visualizations only the activities and the order in which activities happen are visualized. Therefore, currently there is no possibility to show the geographic information and process in the same visualization. Some insights could already be gained by, for example, filtering the data for a process graph to only show information for a particular office and a map with all offices. However, when the combination of the two would be visualized in the same visualization some insights could be gained which otherwise would have been easily missed. For example, the relative distance between two offices could have an impact on a process which is not easy to recognize when the location of a office is displayed as text.

Adding the location of the events will add a new dimension to the visualization. In the context of business processes, the information of at which location an event occurs can help to get more insights into what happens at the different locations. In addition, it could also help to get insights into the cause of bottlenecks in the system. For example, by visualizing a combination of the process and the geographical information, an insight that could be gained is that the cases that have a high throughput time are going through the same office.

1.1 Process mining

The goal of process mining is to get insights about a certain process by interpreting the data associated with it. The data of a process is stored in an event log. Every row in this event log is an event that happens in relation to that process. Every event has an activity associated with it. These activities define what happened during that event. For example, for a process of paying invoices, activities that could happen are: receive invoice, check invoice, and pay invoice. Every event has a timestamp associated with it. These timestamps are used to determine the order in which the events occurred and, when obtaining more detailed insights, to determine the time between activities. The events in the event log are associated with a case. These cases define different instances of a process. In the example of paying an invoice, every invoice is a separate

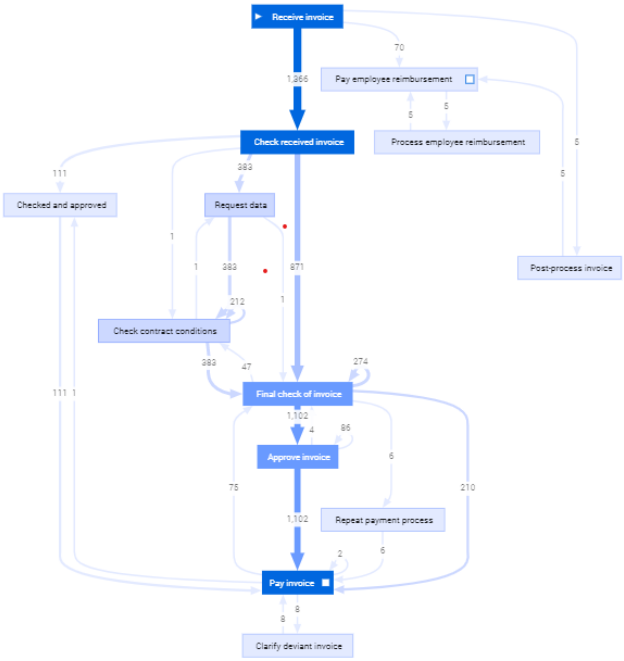


Figure 1: Example of a process model in the UiPath process mining platform [4]. The process model is generated using a directly follows graph and the graph layout algorithm by Mennens et al. [18]

case. To determine the case of an event, every event has a case identifier. This identifier is the same for every event which is in the same case. To use process mining on an event log, the event log should, at least, have the above mentioned activity, timestamp, and case identifier attributes.

Process mining can be used in three different ways: discovery, conformance checking and enhancement [26]. Process discovery takes an event log as input and creates a process model from that event log. Conformance checking takes an event log and process model as input and determines to what degree they describe the same behavior of the process. Enhancement takes an event log and process model as input and extends the process model using information gained from the event log.

There are several possible algorithms that could be used for process discovery. Examples of such algorithms are the inductive miner [15] and the heuristic miner [27]. However, those algorithms have a relatively high running time and generate complex models. Therefore, for most business use cases, a directly follows graph is used as the process model. In a directly follows graph, there is an edge between nodes a and b when event a is directly followed by event b in at least one case in the event log. Figure 1 shows an example of a directly follows graph.

1.2 Graph drawing aesthetics

The process model generated using process mining can be seen as a graph consisting of nodes, which represent the activities, and edges between the activities. Therefore, the resulting graph should adhere to graph drawing aesthetics. Graph drawing aesthetics are characteristics of a graph layout which describe how readable a graph is. Seven of the most common graph drawing aesthetics are [22]:

- minimizing edge crossings,
- minimizing edge bends,
- maximizing symmetry,
- maximizing angular resolution
- maximizing edge orthogonality
- maximizing node orthogonality
- maximizing consistent flow direction (for directed graphs only)

Most of the time it is not possible to optimize an algorithm for a certain graph drawing aesthetic while keeping the other aesthetics the same. For example, when the node positions are changed to maximize node orthogonality, it affects the symmetry because that heavily depends on the node placement. Therefore, most graph drawing algorithm focus on either maximizing the result for one of the aesthetics or find an average result for all aesthetics. In some situations, it is also not possible to influence all metrics. For example, for a graph on a map where the exact location of the nodes is important, it is not possible to change the position of the nodes. Therefore, it is not possible to change the results of the node focused aesthetics.

The above mentioned graph drawing aesthetics are all calculated on static graphs. However, there are also aesthetics that focus on dynamic graphs. Those aesthetics focus on the changes of a graph when one of the parameters changes. This could be the case when part of the data is filtered out or the view of the map changes when the graph is projected on a map. An example of a dynamic graph aesthetic is graph stability. This aesthetic is about minimizing the change in node and edge placement when the input of the graph slightly changes.

1.3 Overlap removal

When all the locations of a particular process are rendered on screen as nodes, it can happen that the nodes will overlap. Overlap between nodes can lower the readability of the graph. Therefore, in most cases, it is better to remove the overlap between nodes. Most of the algorithms to remove overlap do this by either merging overlapping nodes or displacing them [10]. Both of these approaches have advantages and disadvantages. Therefore, the right approach to use depends on the requirements of the given context. For example, when the nodes are drawn on a map and displacement is used to remove the overlap, then it could happen that it is hard to determine the original position of the nodes. When nodes are merged in that example, then it can be difficult to determine where, and how many nodes, are merged in one node. When there are many nodes that need to be shown, it is better to merge overlapping nodes because with too many nodes visible on screen, it can be difficult to derive information from the graph.

1.4 Visualizing geographic information

Geographic information can be visualized in a lot of different ways. Most of these visualizations project the information on a map. Visualizing information projected on a map can give extra insights about the information. For example, the relative distance between points of interest in the data. These visualizations can show information per point or also show relations between points. An example of a visualization which show relations between points are origin-destination flow maps [13]. These visualizations could also be used to visualize a process. However, these visualizations would not visualize important aspects of the process, such as the activities. Therefore, these visualizations are currently not suitable to visualize a process.

1.5 Problem description

Separate visualizations for geographical location information and process mining cannot identify all insights about a process and the geographical information from an event log. Therefore, we want to visualize a process in combination with geographical location information for an event log. From this visualization the process for that event log should be easy to see. In addition, it should be possible to gain insights into how the location of an certain event affects the process. We want to do this by answering the following research question:

How can we visualize the geographical information of a process?

In order to answer this question, we propose a framework for visualizing the geographic information of a process. During creation of this framework, we encountered two other problems for which we needed to find a solution. The first one is how we remove overlap between nodes. With the relatively high number of nodes, we chose to merge overlapping nodes instead of displacing them. In the context of the proposed framework, it would be better to choose the nodes that are merged based on the region in which the nodes are. For example, when a node in The Netherlands is overlapping with a different node in The Netherlands and one in Belgium, it would be better to merge the two nodes in The Netherlands before merging with the node in Belgium. One of the reasons why this is better is that it can help in finding differences of process execution between different regions. For the above mentioned example, suppose the offices in The Netherlands and Belgium execute the same part of a process. However, they have a difference in execution. When The office in The Netherlands is merged with the Office in Belgium it would be harder to find this difference in execution. This results in the following research question:

How can we cluster overlapping nodes using a region hierarchy?

The second problem we want to solve is about routing the edges. In one of the visualizations of the proposed framework, information is rendered inside the nodes. Therefore, it would increase the readability of the information of the nodes, when the edges would not overlap with the nodes. To solve this problem we formulated the following research question:

How can we draw the edges such that there are no node-edge crossings while also minimizing the effect on other graph drawing aesthetics?

This thesis is performed at UiPath Process mining [4], which is part of UiPath. The UiPath process mining platform is a process mining and business intelligence platform used to visualize data using several standard visualization techniques. Examples of the possible visualization techniques are a bar chart or process graph (Figure 1).

1.6 Related work

For both process mining and visualization of location data, a lot of research has been done. However, the combination of the two is not researched often [16]. That thesis also gives a framework for visualizing geographic information of a process. The decision on which visualization from this framework is used partially depends on the type of information the user wants to gain from the visualization. In our research we also have a framework with different visualizations. However, the choice of which visualization should be used is made based on characteristics of the visualized data.

There are visualizations that visualize flows between locations. An example of this are traffic flow visualizations [6], these visualizations show the traffic flow throughout a certain area. These visualizations could also be used to visualize a process. However, they would not visualize important aspects of a process such as the activities.

McNamara et al. [17] introduces a visualization for event-based movement. This visualization shows a ribbon detailing the location of the corresponding person at a certain time. The time someone spends at the same location is visualized as a single block spanning the time the person is at that location. The corresponding location is visualized by a combination of the texture and color of the block. When this would be used for processes, important aspects of process visualizations would not be shown. For example, the process model would not be visualized.

Thudt et al. [25] introduces a visualization for a timeline. In this visualization the timeline is split into different segments in which the person corresponding to that timeline was in the same area. Every segment is represented by a circle showing a map of the area in which the events of this segment occur. Every event is represented by a dot on this map. The size of the circles is determined by the time the corresponding person spent in this area. From the circles alone it can be difficult to determine where in the world this area is. To solve this a map is shown on the bottom which shows where in the world the different circles are. This visualization cannot directly be used for processes as in processes it is most of the time not the case that it is a single sequence of events. Chapter 2 explains how this visualization could be used for one of the categories in the framework.

Krueger et al. [14] introduces a visualization interface to analyze city dynamics. This visualization consists of several parts. The part that is most relevant to process visualization is a map that shows the location and type of events that are happening as text on the location of that event. To avoid clutter of showing too many events, events of the same type that are happening close together are clustered into one event. The size of the text denoting the event type is determined by the number of events of the same type that happen, close to, the same location. There is an arrow between two areas if there were events in the data in which the first event happened in the area at the beginning of the arrow and the event of the same person that is directly after that event happens at the area at the end of the arrow. This visualization could also be used for other processes. However, if there are a lot of different activities on varying locations it can become too cluttered. Chapter 2 describes how this can be used for one of the categories in the framework.

1.7 Overview

In Chapter 2, we propose a framework for visualizing the geographical information of processes. The framework consists of five different visualization techniques. The relation between the location and activity of an event and the relation between location and case determine which of the five visualizations is used.

Chapter 3 describes a novel algorithm which removes node overlap by merging nodes based on the regional hierarchy. This consists of a step that needs to be done once for a given input in which the cluster nodes are pre-computed. The other step clusters overlapping nodes together given a view of the map and a hierarchy.

Chapter 4 describes how the edges are routed such that the edges do not overlap with nodes. This consists of two steps. In the first step we reroute the edges using a visibility graph such that they do not overlap with the nodes. In the second step, we displace the edges along the nodes to lower the number of edge crossings.

In Chapter 5, we evaluate the node clustering and edge routing algorithm using a set of metrics. We selected most of the used metrics from traditional metrics for graph aesthetics. However, we also introduced a metric to measure how much a certain cluster node adheres to a given hierarchy. These metrics are calculated on the calculated graphs of a given set of views of the map. This is done for three different datasets. Chapter 6 summarizes the results and discusses the possible future work.

Chapter 2

Framework

The information that needs to visualized about a process changes depending on the use case of the visualization. For example, for a production process where most locations have a dedicated function, the information relevant to visualize is different compared to a business process where the purpose of the different offices of the company are more loosely defined. In order to give a suitable visualization for such different use cases, we propose four different categories for the processes and a separate visualization technique for the different categories. The four different categories are based on the four cases in the cardinality relationship between the activity and location. This results in the following four categories;

- Many-to-many: In this category activities can happen at different locations and different activities happen at the same location. This category consists of sub-categories. These sub-categories are divided based on the relation between a case and the location of events. The two sub-categories are:
  - One-to-many: Suppose A and B are two events from the same event log which has at least the attributes case id and location. It holds that A.CaseId = B.CaseId $\implies$ A.Location = B.Location.
  - Many-to-many: Events, of data in this sub-category, of the same case can happen at different locations and locations can have events from different cases.

  Note that there could be two more sub-categories for the relation between cases and location. However, for both sub-categories it would be the case that a location would only handle one case. This is not something which would realistically happen nor would it be interesting to visualize this.

- One-to-many. Suppose A and B are two events from the same event log. It holds that: A.Location=B.Location $\implies$ A.Activity=B.Activity.
  Note that there is no distinction for categories where complete cases happen at the same location. Suppose an event log E which has a one-to-many relationship between activity and location and a one-to-many relationship between location and case. That means that every case in E consists of events with the same activity as all events happen at the same location and every event at a certain location has the same activity. There would not be a lot of information that could be gained from visualizing these processes using the geographic context. Therefore, we decided to not consider those sub-categories.

- Many-to-one. Suppose A and B are two events from the same event log which has at least the attributes Activity and Location. It holds that: A.Activity=B.Activity $\implies$ A.Location=B.Location

Note that there is no distinction for categories where complete cases happen at the same location. Suppose an event log E which has a many-to-one relation between activity and location and a one-to many relation between location and case. Then every case which has at least one activity common with another case will happen at the same location because activities only happen at one place in this category. That means that the only cases that happen at different locations are cases that have a disjoint set of activities. In that case it is less relevant to compare locations anymore. Therefore, we do not consider cases happening at one location for this category.

- One-to-one: Suppose A and B are two events from the same event log which has at least the attributes Activity and Location. It holds that:
  A.Activity=B.Activity $\iff$ A.Location=B.Location
  Suppose event log E $\in$ (X activity $\iff$ 1 location) then it holds that E$\in$ (1 activity $\iff$ X location) and E$\in$ (X activity $\iff$ 1 location).
  Note that there is no distinction for categories where complete cases happen at the same location. This is because of the same reason as for the one-to-many category.

2.1 One-to-one

The most important information that can be visualized for the data which is in this category is where the different types of activities happen in relation to the process. However, for some processes more information could be gained from visualizing this than for others. For processes where an object has to physically move between the places, the location of the events influences the duration and the cost of the process. Therefore, visualizing the location of activities could give insights into costs and duration of certain parts of the process. For processes which do not have objects physically moving between locations, it is less useful to look at the geographical information in the context of processes that fit in the one-to-one category. However, there are still some use cases for those processes where it is useful to use this information. An example of information that could be gained from using the geographical information is that one of the offices is in a completely different timezone than the others. In that case, it would be harder to work together with the people in the other offices.

For this category two things are important to visualize, namely: the location of activities and the process graph for the corresponding data. Both aspects should be visualized at the same time as the combination of the two aspects is important to show, for example, the transportation that is needed in a production process.

Visualization
The general idea for visualizing data which is in this category is to map every activity to the location at which it happens on a map. Figure 2 shows an example of this visualization. Activities can overlap because they are too close to each other. In that case, an overlap removal algorithm, such as Dwyer et al. [9], can be used to remove the overlap. It is important that the overlap removal algorithm that is used displaces the nodes instead of clustering the overlapping nodes. Clustering the activities can give the idea that the activities that are clustered happen at the same location while that is not the case in this instance.

The edges that will be drawn in this visualization are based on the edges that are within the process graph corresponding to this data. All the nodes in the visualization represent one of the activities which are also in the process graph so determining between which nodes in the visualization an edge needs to be drawn is trivial. To make sure the edges do not overlap with the nodes, the edge drawing algorithm by Dokulil et al. [8] can be used.

In general, basing the position of an activity on the location works well. However, suppose there

(a) (b)

Figure 2: The visualization of the one-to-one category. a) The map on which the activities are mapped. b) The corresponding process graph. The activity with the bold border is the start-activity of the process. The activity with the dashed border is the end-activity

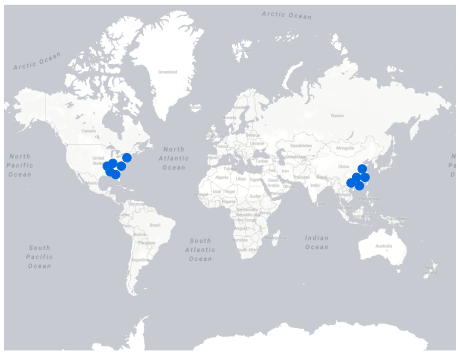


Figure 3: Visual example of how locations of activities could be split up in two groups. The blue dots are the locations of the activities.

are two groups of activities. In those two groups, the activities are close together. However, the two groups are far apart. Figure 3 shows this example. In that case, this visualization cannot show the interaction between activities correctly. Zooming in will also not fix this problem as in that case only one group is shown at the same time. To fix this, the position of the activity should not fully depend on the location of that activity. The position of the activity could partially depend on the position in the layout of the process graph. The question is then how much should the position depend on the process graph and how much on the location? This varies per use case: when the location of the activities is equally distributed over a certain area, then it is not necessary to let the position depend on the process graph. While when the activities are clustered in groups, the position should depend heavily on the process graph. Therefore, the ratio between using location versus position in the graph is different per use case. Therefore, there is not one ratio for determining position which works for most use cases. Therefore, it should be possible to have different ratios for the different use cases. One way to do this is to use a force-directed layout algorithm in combination with a slider similar to the visualization described in Saffo et al. [23]. In this paper a force-directed layout algorithm is described which determines the position of a node by using the location and the position in a graph with a ratio between them chosen by the user

using a slider. In this paper the network graph is a graph displaying co-authorship network. This graph could also be replaced by a process graph and then it would work for this visualization.

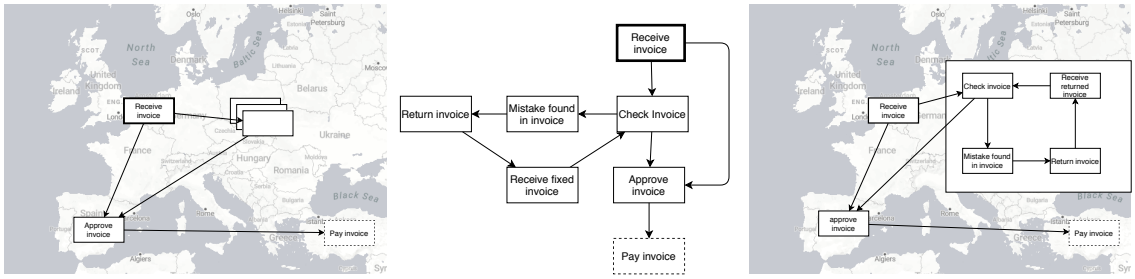
The position in the process graph depends on the relationships with the other nodes in the process graph. This could change when one of the nodes is not shown anymore. Basing the position of nodes on the geographical locations is, therefore, more stable than basing it on the position in the process graph. Therefore, basing the position of an activity partially on the position in the process graph can decrease the stability of the graph. How much the stability of the graph is changed depends on what kind of layout algorithm is used to determine the layout of the process graph. One of the algorithms that already creates a stable layout is the algorithm by Mennens et al. [18]. The change in stability also depends on the ratio between the dependency of the position on the location and on the position in the process graph.

2.2 Many-to-one

In this category, multiple activities can happen at the same location but every activity only happens at one location. For this category, it is useful to visualize at which locations certain aspects of a process takes place. This information is useful to determine which parts of a process could be offloaded to a different location if that location is overworked or to move parts of the process to decrease the (information) transportation needed. Therefore, it is not only useful to visualize processes with a physical product moving between the locations of the activities, but also for other types of processes.

In the visualization for this category, it is important to show the different locations at which activities happen and what activities happen at a certain location. In addition, edges should be drawn between activities in the case there is also an edge between those activities in the process graph corresponding to the data. The information that needs to be visualized is similar to that of the category described in Section 2.1. Therefore, a similar visualization would also work for this category.

Visualization
The visualization that is used for this category is based on the visualization described in Section 2.1. This visualization, however, does not work well for activities that happen at the same location as other activities. When there are multiple activities at the same location, this visualization will try to place those activities at the same location. This will result in overlap between the locations. By the overlap removal algorithm, they will be displaced such that they will not overlap. This will result in possibly several activities directly next to each other which makes it hard to see the edges between the activities. This is already partially fixed by letting the position of an activity be based more on the position in the process graph. However, this has as downside that the information that an activity happens at the exact same location is then lost. To fix this all the activities that happen at the same location are clustered into one block and then the visualization is shown using one block per location. Clicking on this block will show the activities that happen at that location. The edges between activities in that block and edges between a different block and an activity in this block are also shown. Figure 4 shows an example of how this visualization could look.



(a) (b) (c)

Figure 4: a) The map where the (clusters of) activities are mapped on. b) The process graph without clusters. c) The map when a cluster is selected. All the edges that originally went to/from the cluster now go to/from the actual activity. The activity with the bold border is the start-activity of the process. The activity with the dashed border is the end-activity

The slider for the ratio still uses the layout of the process graph as one of the aspects, however, a standard process graph of the data will not work as that could split activities which happen at the same location. Splitting those activities would remove the information of which activities happen at the same location. First, a process graph is made based on the data using a chosen process mining algorithm. After that, the activities that happen at the same location are clustered and the edge between the clusters are based on whether there is an edge between one of the activities of both clusters. This will result in a new graph on which a graph layout algorithm, such as the algorithm by Menners et al. [18], can be used to determine the layout.

2.3 One-to-many

In this category, activities happen at multiple locations but at a certain location only one activity happens. For this category, it is useful to visualize the locations of the different activities in combination the process graph. This information could be used to determine whether cases are always routed through the right locations to handle the activities. This information is more important for processes where a physical object moves between the different locations. For other processes, the information is still useful as this can be used to determine if two locations should be merged. Information that could be gained from this visualization that could help make that decision are: how close together the locations are and similarity in cases that are handled by the locations.

We cannot use the visualization of the two categories above for this category. That visualization has one node per activity, while in this category, an activity can happen at multiple locations. This could be integrated by making a separate node per location of an activity. However, when activities happen at a lot of different locations, this can lead to a lot of different nodes on the screen, which makes it hard to understand the visualization. Additionally, it is desired that it is easy to determine which activity happens at a location and determine all the locations where that activity happens. With the locations of all the activities in one map, it is hard to do that. It could be easier to visualize that by assigning a color to an activity and color the nodes the corresponding color. However, in that case, it would still be hard to determine all of the locations of one particular activity.

Another technique to visualize this is to use a visualization for origin-destination for a specific pair of activities. An example of such a visualization is Flowstrates by Boyandin et al. [5] With this visualization, it is possible to show the location of the activities in the case where the two activities happen directly after each other. By only showing the locations of the events where

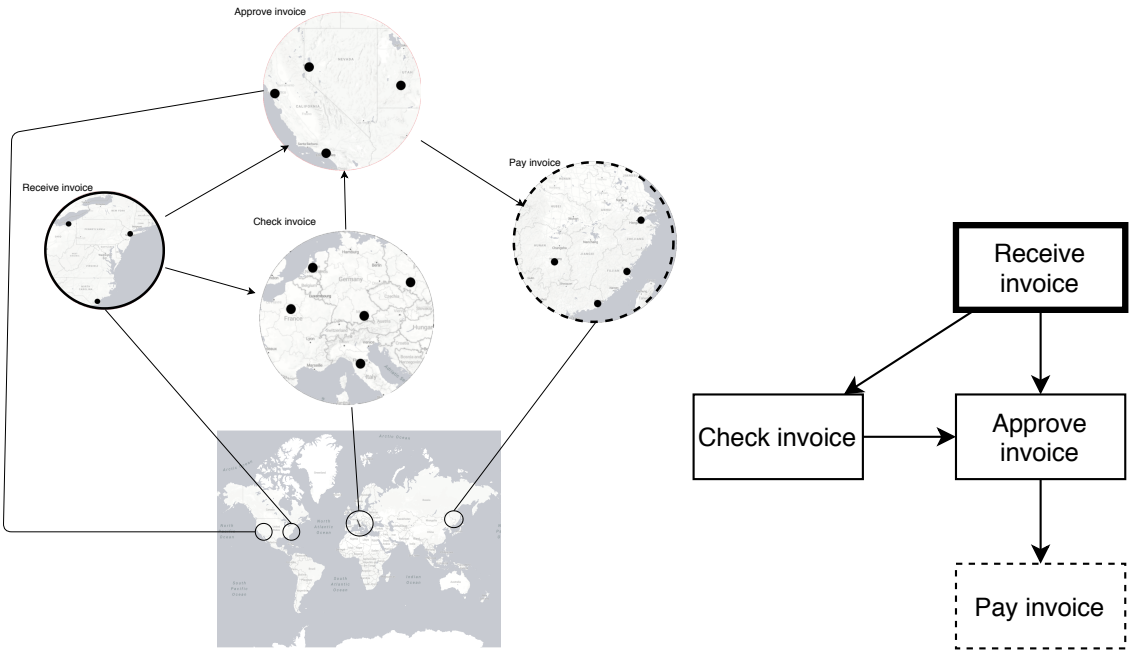
the two activities happen directly after each other, it can be the case that not all locations of an activity are shown in this visualization. This visualization will also not show what happens before or after the selected activities, which makes it harder to analyze the location of an activity in the context of the complete process. Therefore, this visualization does not show the complete information for data that fits in this category. It can, however, be used in combination with a visualization which shows the complete context of the process to analyze a specific edge between two activities.

Visualization

The visualization that is used for this category is based on the Visits visualization from Thudt et al. [25]. This paper describes a spatiotemporal visualization which shows the order of events of visiting places. This visualization can be modified to show a process by showing a circle per activity, instead of a circle per region. The map inside the circle will show all the locations where the corresponding activity happens. In addition, edges between activities can be added based on if there is an edge between the two activities in the process graph. The placement of the circles and edges depends on the chosen layout algorithm that is used on the process graph. The size of the different circles depend on the number of different locations at which the activity happens. This is to make it easier to look at where an activity occurs in cases when this is at a lot of different locations. It can happen that an activity happens at very widespread locations. In that case it can be hard to look where a specific location of that activity is exactly. To fix this it would be possible to open a big map only showing the places where the corresponding activity happens by clicking on the appropriate circle. Figure 5 shows an example of this visualization.

When the places that an activity occurs are close to each other, it can be the case that it is hard to determine where exactly in the world those activities happen. To make this more clear, the Visits visualization shows a bounding box on the map for every circle to show the location in the context of the activities. Which bounding box belongs to which circle is shown by two lines between the circle and the bounding box. When there are a lot of activities for which circles are drawn, it can be hard to determine which bounding box belongs to which circle, especially when the bounding boxes are overlapping. One of the ways to fix this is to show every bounding box in a different color. Or by highlighting the bounding box for a circle if that circle is selected.

In this visualization, it is not possible to see, for an edge in the process graph, between which combination of locations that edge occurred. As the locations in the circle for a particular activity only imply that at that location the activity occurs and nothing about what happens before or after. One way to fix this is by only showing the circles of the locations of the two activities between which the selected edge is. In that case in the circles only the locations are shown which have an edge to or from the other activity. This will, however, still not show the complete information for that edge because it is not between which locations an edge occurred. To fix this a visualization like Flowstrates, as described above, can be used to show more information for a particular edge in the process graph.



(a) (b)

Figure 5: a) The map on which the activities are mapped. Based on the Visits [25] visualization b) The process graph used in the visualization in a. The activity with the bold border is the start-activity of the process. The activity with the dashed border is the end-activity

2.4 Many-to-many

For this category, there is a difference when cases happen at the same location or not. Section 2.4.2 describes the visualization when the complete case happens at the same location. Section 2.4.1 describes the visualization when this is not the case.

2.4.1 Case at different locations

In this category, activities can happen at different locations and at a certain location different activities can happen. This category is a combination of the two many-to-one categories described above. Therefore, the information that is important to visualize in this category is also a combination of those two categories. This results in that it is important to visualize the different aspects of a process that happen at a certain location, the interaction between different locations, the locations at which a certain activity happens and an overview of the complete process graph.

A visualization that could be used for this is the visualization that is described in Section 2.2. This visualization will display one block per location. In the case of the many-to-one relation this is at most the same as the number of activities. Which, in general, will result in a visualization with a number of blocks on screen for which it is still easy to understand the complete graph. For the category described in this section, the number of different locations does not depend on the number of activities. This results in a uncertainty on the number of blocks in the visualization. To make sure the resulting visualization shows a reasonable number of blocks some form of clustering is needed. However, the combination of clustering blocks in combination with the ratio between position based on location and process graph will lead to a visualization that is harder understand.

When the ratio on the slider change, the position of the nodes change. Therefore, the nodes that are clustered would change. This change in clustered nodes makes it harder to understand the visualization. Another possibility is to remove the possibility to let the position depend on the position in the process graph. In that case only clustering is used to make sure the nodes on the final visualization are still easy to understand. However, in that case, only showing a block with the activities that happen for all nodes in that cluster is not enough information for this visualization to be useful.

Another possible visualization is the visualization that is described in Section 2.3. This shows the distribution of where activities happen very well. However, by showing the locations per activity it is hard to see all the activities that happen at a certain location. To find this information you have to look at the circles of all the activities to check if that activity happens at that specific location. This makes the Visits visualization not useful for this category. When using this visualization, the data could also be easily misinterpreted by thinking that two activities happen at the same location while they actually happen at two different locations close to each other. A way to fix this is to make it possible to highlight a location in one of the circles, after which the same location is highlighted in the circles of the activities that also happen at that location. This makes it easier to see which activities happen at the same location. However, in that case it is still hard to find the interaction with other locations for a certain location.

Another visualization that could be used is based on a visualization of Krueger et al. [14]. This visualization is designed to visualize mobility patterns. Which has similarities with visualizing a process. Therefore, this can also be used to visualize processes. The standard map shows the activity as text for every event at that location. When there are multiple of the same activity they are clustered. This is not the best visualization when there are multiple activities happening at the same location. It is also possible to show a radial area chart overlay instead. A modified version of this can be used to show the distribution of the activities that happen at that location.

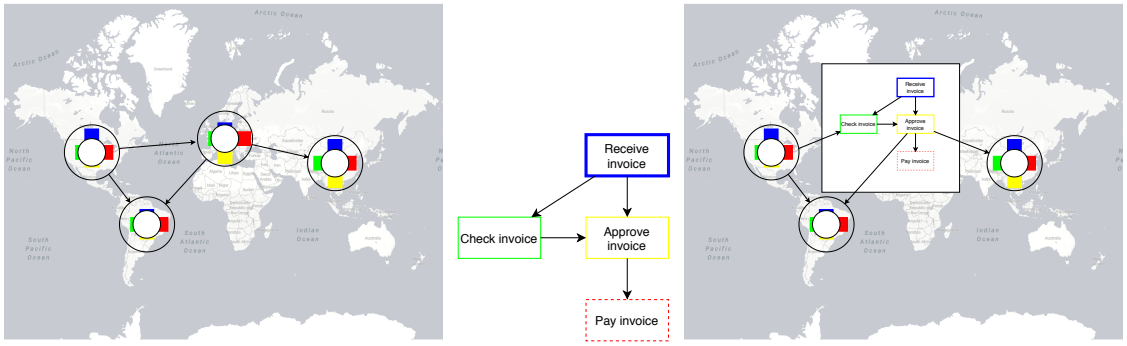
Visualization

For this visualization for every location at which activities happen, a circle is drawn on the map similar to the radial area chart as described in the paper. On this circle extra information about that location could be shown. An example of information that could be shown in this circle is the distribution of how often a certain activity happens at that location. An edge between two locations is drawn if there is an edge in the process graph between an activity occurring at the first location and an activity occurring at the second location. Overlap between circles will be removed by clustering. When two or more circles are clustered all incoming and outgoing edges will be changed to start or end at the new clustered node. Figure 6 shows an example of this visualization.

Clicking on a circle will show a process graph for the corresponding location or cluster of locations. This can be used to determine which parts of the process happens at a certain location and how that part differs from when it happens at a different locations. The process graph will also include edges going to, or coming from, other locations.

Zooming in will make it possible to look at the interaction between different nodes that are clustered on the complete map. To make it possible to still look at the interaction with other clusters, the clusters outside the zoomed-in view will be mapped to the border of the frame if they have an edge to or from one of the clusters inside the frame. The level of clustering of the nodes on the border will be the same as on the overview level where all the locations are visible.

Mining the process graph has to be done slightly different. Normally the process graph would contain a single node per activity. For this visualization, however, this would not give enough information. Because the visualization shows an edge between locations, it needs to be shown if there is an edge between two activities if those activities happen at a certain location. To fix this a different activity, which has different values for every possible location an activity can occur, is



(a) (b) (c)

Figure 6: a) The map on which the activities are mapped. Based on the visualization by Krueger et al. [14] b) The process graph used in the visualization. The activity with the bold border is the start-activity of the process. The activity with the dashed border is the end-activity. c) The visualization when the node in Europe is selected.

used to determine the process graph. After that all the activities that happen at the same location are clustered into a single node.

One of the disadvantages of this visualization is that the complete process graph is not visible. Only the interaction between different locations are shown. It is not possible to also show the process graph in the same visualization. Therefore it should always be used in combination with a standard process graph.

Clustering

For the clustering of the nodes, a context-aware clustering algorithm is used. Context-oblivious clustering algorithms do not give the best result for this visualization because context of the nodes that are clustered is important. For example, it would be better to first try to cluster nodes which are in the same country instead of clustering with a node in a different country. One of the reasons for this is to make it easier to identify differences between regions. For example, suppose there are two regions which have a difference in execution of a process. When one node of from each of those regions is merged, the process graph corresponding to that cluster node will show both executions of the process without the possibility to see that one execution only happens at a certain region. When the nodes from the same region are clustered , the corresponding process graph will only show the execution of the process for that region.

To determine which nodes belong to the same region a hierarchy tree is used. This hierarchy tree consists of several regional levels (such as state, country, and continent). using the values of each level a tree of hierarchies can be build. The nodes which are clustered to remove overlap between two nodes is determined by the position of those two locations in the hierarchy tree. For this to work, a complete hierarchy is not needed, only the nodes in the hierarchy tree that has a location at which an event occurs as leaf needs to be available. Chapter 3 describes this in more detail.

Edge drawing

The shown nodes display process information about the nodes that are clustered in that node. When edges would overlap with those nodes, the readability of the information would decrease. This could be fixed by drawing the edges underneath the nodes. However, this would make it harder to follow the edges through the graph. Therefore, we decided to use an edge drawing algorithm which completely removes the overlap between nodes and edges. Chapter 4 describes this in more detail.

2.4.2 Complete case at same location

In this category there is no interaction between different locations as a complete case always happens at the same location. Therefore, there is not a lot of information that could be gained from plotting the different parts of the process on a map as there is no interaction between different locations. However, in this category it is more important to show the differences between the cases of the process that happen at the different locations. It could still be useful to have the possibility to show a map of the places where cases of the process are handled. This could be used to find a reason why there is a difference between cases of different locations. An example of a difference that could be found using this map is if there is a group of places that have the same difference in the cases of the process and all those locations are in the same country. In that case that could be the reason for the difference in the cases in those places.

The visualization for the many-to-one category could work for this category as well. This will show a node per location, without edges between nodes as there is no case that happen at multiple locations. Interacting with a node will show the process graph for the process occurring at that location. Allowing to show the process graphs of multiple locations at once will allow to compare the process graphs of different locations. However it is hard to spot small differences, especially for complex processes, when comparing different process graphs in separate graphs. It is easier to see differences when the process graphs are merged into one process graph with some way to highlight differences of the locations.

Visualization
A visualization that can be used for this category is KelpFusion [20]. KelpFusion is a visualization technique for visualizing different sets of items on a map. This can also be used for processes because the process graph for a particular location can be seen as a set of nodes and edges. Therefore, every location will have a set of nodes and edges and then KelpFusion will show these different sets in one visualization. This will make it possible to compare the different locations in one visualization. Visualizing the process graphs of the different locations in this way will allow the user to see the differences between the executions of the process at different locations. This will not show where the different locations of the process executions are. To show this, a map can be added which shows a marker for every location. To identify which set of nodes and edges belongs to the same location, the markers on the map and sets of nodes and edges will have the same color for the same location. In the case that two locations have the same set of nodes and edges they will also have the same color to make it easier to show that they have the same process graph.

One downside of this approach that it does not work well when there are a lot of different locations with different process graphs. In that case, it can be hard to determine which nodes and edges are in the process graph of a particular location. One way to fix this is to add the possibility to highlight a certain location. In that case, the location is highlighted on the map and all the nodes and edges of the process graph for that location are highlighted in the complete process graph. To make it possible to still compare different locations in that case it is possible to highlight multiple locations.

Chapter 3

Node clustering

In this chapter, we propose a novel context-aware clustering algorithm which clusters nodes based on a given hierarchy. The nodes that are clustered are chosen based on the position of the nodes in the hierarchical tree.

This algorithm consists of three steps. The first step is precomputing a cluster node for all the nodes in the hierarchy tree (Section 3.2). After that the nodes that are currently visible are selected (Section 3.3). These selected nodes are then clustered in the last step until there is no overlap anymore (Section 3.4). When the nodes are merged the incoming and outgoing edges of the nodes are also merged or relocated to the merged node. Chapter 4 describes how we obtain these edges. Figure 7 visualizes the different steps as a graph. The solid arrows are steps which are described in this chapter.

We implemented a prototype of this algorithm as an extension of a visualization template in the UiPath Process Mining platform [4]. Examples of such templates are a process graph or a bar chart. The template which is used as the base for the implementation described below is a template to visualize data points on a map. A library called Leaflet [3] is used to show a map in the template and to facilitate the possibility to draw basic shapes on the map. The platform is based on a client-server architecture and this is also utilized in the prototype. The dataset that is used for this prototype is described in Section 3.1.

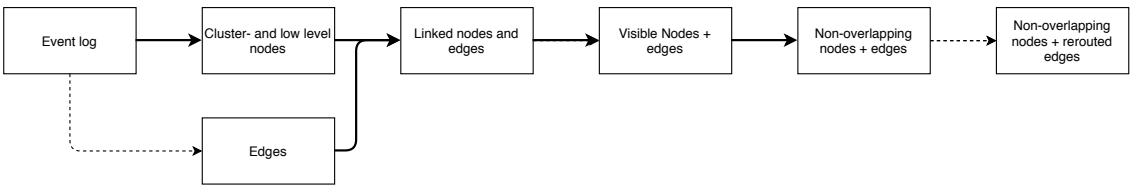


Figure 7: Figure of the different steps needed to compute the drawn graph based on an event log. The solid arrows are steps described in this chapter. The dashed arrows are steps described in Chapter 4

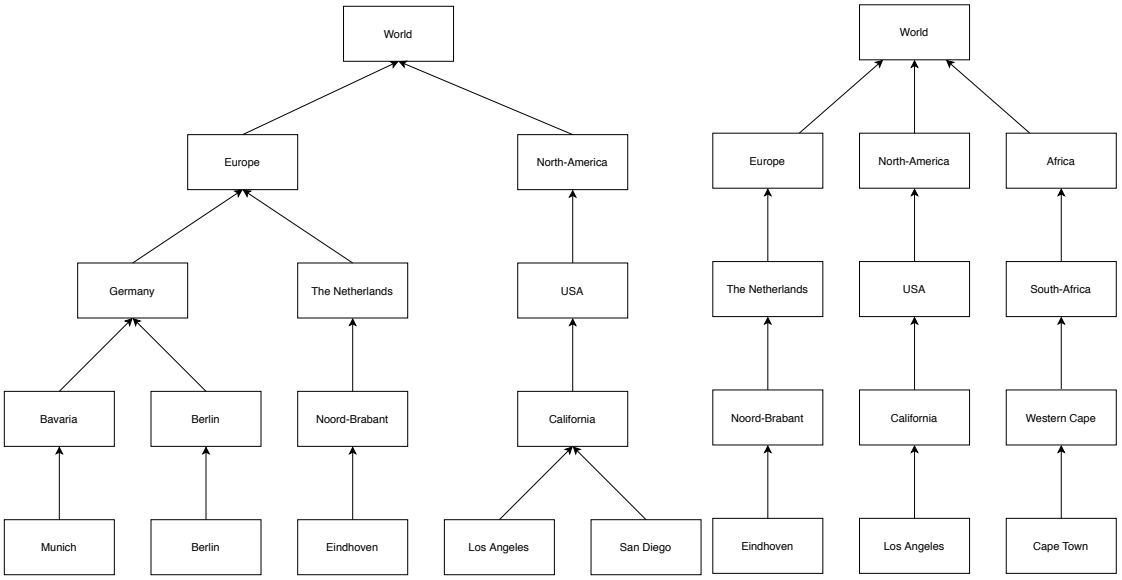


Figure 8: Examples of region hierarchy trees that could be used in the clustering of overlapping nodes. b) Example of a hierarchy tree with the maximum number of cluster nodes.

3.1 Input

This clustering algorithm is used in the context of process mining. When we create the leaf nodes and cluster nodes, some information about the events is used. Examples of this are the distribution of the activities between all the events that happen at a certain location Therefore, the input of this algorithm is an event log instead of a set of points that needs to be visualized. We assumed that the events have separate hierarchy level attributes for every hierarchy level in the used tree. All those hierarchy level attributes are needed to create the tree that determines the cluster nodes. For all those hierarchy attributes, the geographical location should also be known. In addition, the events should also have a separate attribute of the actual location at which the event happened. The events also need to contain the attributes needed to do process mining. These attributes are: activity, timestamp, and case identifier.

3.2 Hierarchy

The algorithm starts off by creating a hierarchy tree. Every event has attributes associated with it that correspond to the different levels of region hierarchy it belongs to. An example of this could be Eindhoven, Noord-Brabant, Netherlands, Europe, and World for an office in Eindhoven. The hierarchy tree is created from these hierarchy values. Figure 8 shows an example of a hierarchy tree. The hierarchy tree is created using a built-in feature of the UiPath Process-mining platform.

The leaves in this tree are all the places where events happen. It is assumed that the location of the events is already the same for the events that should be clustered on the lowest level. For every node in this tree, a cluster is precomputed of all the leaf nodes that are in the tree with this hierarchical node as the root node. These are precomputed as they are often used in the clustering algorithm described below and they do not change unless the used data is changed.

Algorithm
Algorithm 1 describes how the hierarchy nodes are created. The initial call to this algorithm is done with the root of the hierarchy tree as the parentNode parameter. When the algorithm is done, there are cluster nodes for all nodes in the hierarchy. These cluster nodes contain information used in the visualization of the nodes. For example, the number of events per activity for all leaf nodes in this cluster node. In addition, the cluster node also has a predefined location determined by the node in the hierarchy. These cluster nodes are created by recursively going through all nodes in the hierarchy and creating a cluster node with the appropriate location and other information. The base case in this recursion is the case when a leaf node in the hierarchy is found. As described above, the leaves of the tree are all the places at which events occur. For the example of activity counts, the number of events per activity are counted for every location and are added as the activity counts per activity to the leaf nodes in the new tree. For every cluster node, the number of events per activity of all the children in the hierarchy are added together and added as information to the cluster node. The leaf nodes are linked to the edges in the process graph. Section 4.1 describes how the process graph is calculated.

Algorithm 1 createHierarchyNodes(parentNode)

```
if isLeaf(childNode) then
    leafNode = createLeafNode(leafNode)
    return leafNode
else
    initialize clusterNode
    for c ∈ parentNode.children do
        childNode = createClusterNode(c)
        add childNode information to clusterNode
    end for
    return clusterNode
end if
```

Running time
Creating the leaf nodes depends on the number of different activities that happen at that location. In the worst case, all the activities happen at every location. Therefore, the total running time to create all the leaf nodes is $O(n * a)$ where n is the number of leaf nodes and a is the number of different activities. There is one step for the cluster nodes for which every child node needs to be checked. Every child nodes has to be iterated over once. Therefore, for the whole graph, this would have a running time equal to the total number of child nodes. This is $O(n + c)$ where c is the number of cluster nodes and n is the number of leaf nodes. The remaining of the steps for the cluster nodes can be done in constant time. Therefore the running time of creating all cluster nodes is $O(n + c)$. In the worst-case the total number of cluster nodes is $O(n * l)$ where n is the number of leaf nodes and l the number of hierarchical levels. This is the case when the root node of the hierarchy has n child nodes and all of those child nodes have one child node. In that case, also every cluster node in the other hierarchical levels has only one child node. Figure 8b shows this example. This will make the running time for creating the cluster nodes $O(n * l)$. This, combined with the running time of creating the leaf nodes, makes the total running time $O(n * a + n * l)$.

3.3 Sending to client

This part and the part described above are all performed on the server. The parts described above are exclusively executed when the data changes (this can be a data refresh or a filter being

applied). The information that is sent to the client depends on the current longitude and latitude bounds between which all points and edges are currently visible.

The cluster nodes that are sent to the client do not have information about the incoming or outgoing edges. If the nodes would have that information, then every time that node is sent to the client it needs to be checked that the other endpoint of that edge is also inside the current view of the map. If that is not the case, then the edge should be removed from that node. To do this, we would need to iterate over all the incoming and outgoing edges of every node. Every edge would be stored at a node in every hierarchy level. Therefore, every edge needs to be checked on every hierarchy level. Instead, we only store the incoming and outgoing nodes at the leaf nodes and the edges are added to the cluster nodes dynamically. Therefore, every edge only has to be checked once. The edges that are sent to the client are all the edges which are between two leaf nodes which are within the current view of the map. On the client side, the edges are merged or moved to the appropriate node when one the endpoints of the edge is merged.

Algorithm

Algorithm 2 describes the algorithm used to select the nodes and edges. The input of this algorithm is a list of all leaf nodes in the hierarchy and a list of all edges. When nodes are checked to be visible, it is checked if they are inside the current view of the map. We check this for every leaf node. If that node is inside the current view, then that node, together with all the parent nodes in the hierarchy up until the root node, are sent to the client. For all the edges, the algorithm checks that both endpoints of that edge are inside the view. If that is the case, then that edge is also sent to the client.

Algorithm 2 SelectNodesEdges(lowLevelNodes, edges)

```
for n ∈ lowLevelNodes do
    if isVisibleInView(n) then
        n.isVisible = true
        add n to visiblePoints
        while hasParentNode(n) & !n.parentNode.isVisible do
            n = n.parentNode
            n.isVisible = true
            add n to visiblePoints
        end while
    end if
end for
for e ∈ edges do
    if e.fromNode.isVisible & e.toNode.isVisible then
        add e to visibleEdges
        e.isVisible = true
    end if
end for
```

Running time

Checking whether a node is inside the view can be done in constant time. In the worst-case all parent nodes need to be checked to be added to the set of points that are sent to the client for all visible points. This happens when the root node has a number of child nodes which is the same as the total number of leaf nodes (see Figure 8b). In that case, for all visible leaf nodes, all parent nodes need to be set to visible as they cannot have been set to visible by another parent node as that leaf node is the only node that has those nodes as parent nodes. In the worst case all leaf nodes are visible. Adding the node to the list of visible points can be done in constant time.

Therefore, the total running time is $O(n * l)$ where l is the number of hierarchy levels and n is the number of leaf nodes.

Checking whether both endpoints of an edge are visible can be done in constant time. Adding the visible edge to the list of visible edges can be done in constant time. So the running time for this part is $O(e)$ where e is the total number of edges. So the total running time is $O(n * l + e)$.

3.4 Clustering

Algorithm 3 describes how all the overlap is removed. Choosing which nodes should be clustered together is first determined by finding the pair of nodes that have the highest overlap. When we remove the overlap in the order of nodes with the highest overlap, the resulting clustering does not depend on the order in which the nodes are in the list but only on which nodes are in that list. Therefore, the stability of the graph increases. The nodes that are checked for this are the nodes that are currently visible on screen. The value of the overlap that is used for this is defined as the penetration distance as described in the paper by Scheepens et al. [24]. In the beginning all the leaf nodes are set to be visible, all hierarchy nodes are set to be not visible. The nodes with the biggest overlap will not just be merged together to fix the overlap, the algorithm described in Algorithm 4 is used to determine which nodes should be clustered together in order to fix the overlap. That clustering algorithm will also make sure the appropriate nodes are set to be visible for the next iteration of overlap check. Note that during each iteration of the clustering algorithm at least two nodes are merged together into one node. This means that every iteration of the while loop starts with less nodes visible than the iteration before. Therefore, the algorithm will always reach the state where there is only one visible node left, if during all the other iterations there was still overlap between nodes. When there is only one node visible there is trivially no overlap. Therefore, the algorithm will always end in a state where there is no overlap between the visible nodes.

Algorithm 3 clustering(nodes)

```
while there are overlapping nodes do
    find pair of nodes i,j with highest overlap.
    clustering(i, j)
end while
```

For the part where the overlap between two nodes is removed, we first check whether both nodes have the same parent node. If that is the case they are merged together. This is visualized as step 5 in Figure 9 and in Figure 10 in combination with the shown view. Merging the nodes in this case is almost the same as described in Section 3.2. In addition the two nodes are set to be not visible and the merged node is set to be visible. This is used in the next iteration of checking which nodes should be merged. Also, for the edges which are going to or coming from either of the nodes which are merged, the appropriate endpoint of the edge is changed to the merged node. If both points have an edge going to or coming from the same point, the edges are merged into one edge.

If the two nodes do not have the same parent node, then all sibling nodes of both nodes are checked if merging with that node would remove the overlap. If such a sibling is found, then the two nodes are merged together in the same way as described above. This is visualized as step 1 and 3 in Figure 9. If multiple siblings would remove the overlap, the sibling closest to the original node is chosen to be used. This way the order in which the sibling nodes are stored does not have an influence on the chosen sibling for merging. This results in an increased stability of the created graph. Only the siblings that are currently visible are checked to fix the overlap.

In the case that there is no sibling node found which removes the overlap, the node with the lowest

current hierarchical level will be replaced by the parent node. This is visualized as step 4 in Figure 9 All nodes which have that node in their line of nodes to the root node will be replaced by that node even if they don't have that node as their direct parent node. The node for that parent node is already precomputed on the server side as described in Section 3.2. The only part that is left to be done on the client side is linking the edges of all the node that are replaced by the cluster node to that cluster node.

If both nodes are on the same hierarchical level, then we select the parent node of the node with the lowest number of sibling nodes to be used. This is visualized by step 2 in Figure 9 This will reduce the number of nodes that need to be merged. Additionally, in this case, the chosen nodes to be clustered is not dependent on the order of the nodes in the parameters in almost all cases. Only when both the hierarchy level and the number of siblings for two overlapping nodes is the same, then the chosen node is influenced by the order in which they are in the input.

Algorithm 4 removeOverlap(node1, node2)

```
if node1.parent = node2.parent then
    mergeNodes(node1, node2)
else
    Find closest neighbour i of either node which fixes the overlap by merging the nodes
    if No such neighbour exists then
        if node1.hierarchyLevel ≤ node2.hierarchyLevel then
            useClusterNode(node2.parentNode)
            if areOverlapping(node1, node2.parentNode) then
                removeOverlap(node1, node2.parentNode)
            end if
        else
            useClusterNode(node1.parentNode)
            if areOverlapping(node2, node1.parentNode) then
                removeOverlap(node2, node1.parentNode)
            end if
        end if
    else
        mergeNode(Node, i)
    end if
end if
```

Running time

In the worst case, during every iteration in clustering, only one pair of nodes is merged. In that case, the total iterations of the while loop in the algorithm is equal to the total to the number of visible points in the beginning, which is equal to the number of leaf nodes currently visible. For every iteration, all pairs of visible points need to be checked for overlap and the size of the overlap. This check can be done in constant time. Therefore, the running time for this part is $O(n^3 * c)$ where n is the total number of leaf nodes and c is the running time of the algorithm to remove the overlap between 2 nodes.

Almost every step in the removeOverlap algorithm can be done in constant time. However, there is one step that checks all the siblings of a given node. The worst-case running time for this part is $O(n)$. This is the case when the number of siblings is equal to the total number of leaf nodes minus one. Note that when this is the case, for all other nodes in the hierarchy, the number of child nodes is equal to one. Because by having one node with a number of child nodes equal to the number of leaf nodes it means that the other nodes cannot have more than one child node. If there would be a node with the number of child nodes higher than one, then the number of leaf nodes increases. This makes it that it is not equal to the number of child nodes of the node

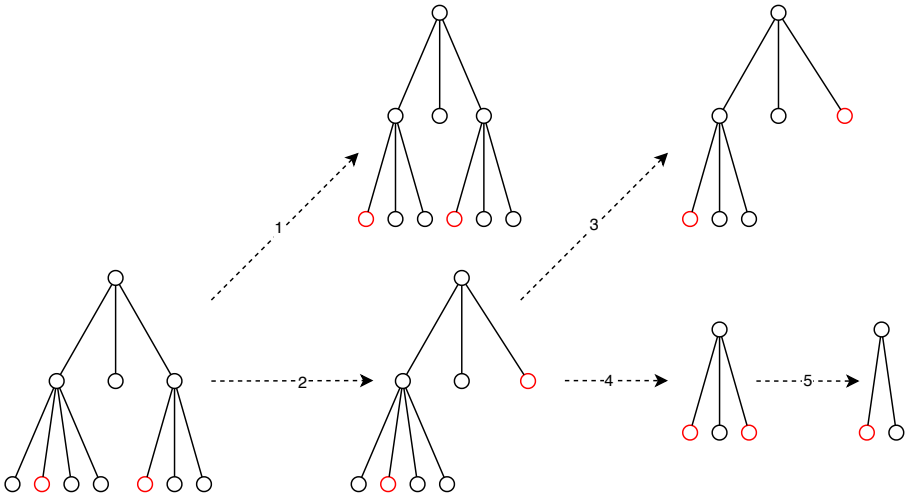
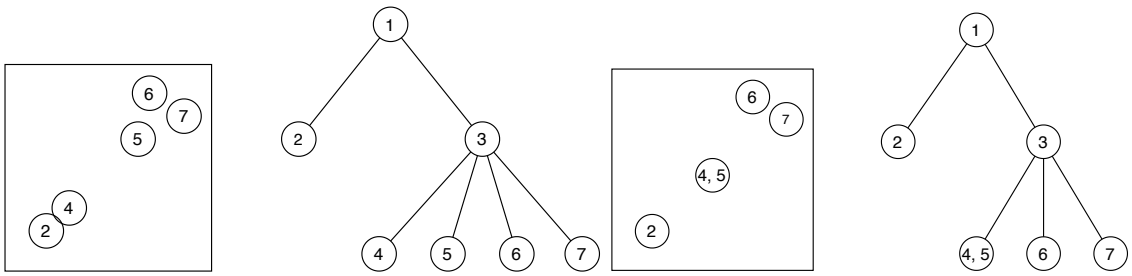
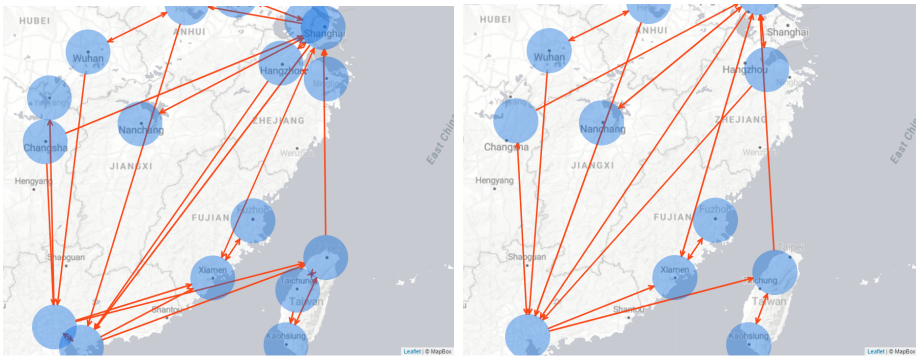


Figure 9: Visual example of how the overlap between two red nodes in the left most tree can be removed using the algorithm described in Algorithm 4. The numbers of the edges represent in which order the steps are executed. For step 1 and 3, it is checked whether merging with the closest sibling solves the overlap. If that is the case, the algorithm is done, otherwise the algorithm will go back to the tree before merging and go further with step 2 or 4 respectively. The red nodes denote the input nodes for which the overlap needs to be removed.



(a) (b)

Figure 10: Example of how the overlap is removed by the algorithm. The squares represent the view of a map. The circles inside are the points currently visible. The numbers inside the circles correspond to a node in the hierarchy. a) The view before the overlap between point 4 and 2 is removed. b) The view after the overlap is removed by merging point 4 and 5.



(a) (b)

Figure 11: Comparison of the overall graph before the clustering algorithm is used (a) and after it is used (b).

with the highest number of child nodes in this tree. This should not be the case in this example. However, it can happen that one of the input nodes, for every time this algorithm is called, is one of the nodes with $n-1$ siblings. Therefore the total running time of this algorithm is $O(n)$ This results in a total running time of the clustering algorithm to $O(n^4)$.

3.5 Conclusion

Figure 11 shows a graph before and after the nodes are clustered together using the algorithm described above. The overall running time of this algorithm is $O(n*a+n*l+e+n^4)$. Where n is the number of leaf nodes, a is the number of activities, l is the number of hierarchical levels and e is the number of edge. In almost all cases n^4 is the biggest factor in this algorithm. The step corresponding to the running time of $O(n^4)$ is one of the steps that needs to be done every time the view of the map changes. To keep the map responsive in use it should not take more than a few seconds to compute the clustering. With the high polynomial running time of the clustering increases quickly when the input size increases. However, in most use cases of this algorithm the number of leaf nodes is in the order of hundreds, as it represents offices of a company. In those cases the running time does not have a significant influence on the responsiveness of the map. However, for uses cases with a larger number of leaf nodes it could be useful to improve this algorithm by, for example, using the visible nodes before the view changes to calculate the visible nodes in the new view.

Chapter 4

Edge routing

In this chapter we propose an edge routing algorithm based on an approach used for robot motion planning [28]. This algorithm creates the edges by finding the shortest path through a tangent visibility graph. Our algorithm is used in the context of process mining. Therefore, the input of this algorithm is an event log instead of a set nodes and edges.

The algorithm consists of three parts. The first parts creates the edge by calculating the process graph (Section 4.1). The second part routes the edges through the graph without intersecting with the nodes (Section 4.2). After the edges are rerouted, the edges are displaced around the circle to decrease the number of edge crossings around the circle (Section 4.3) . This algorithm is implemented in the prototype described in Chapter 3. Figure 12 shows the different steps executed by this algorithm.

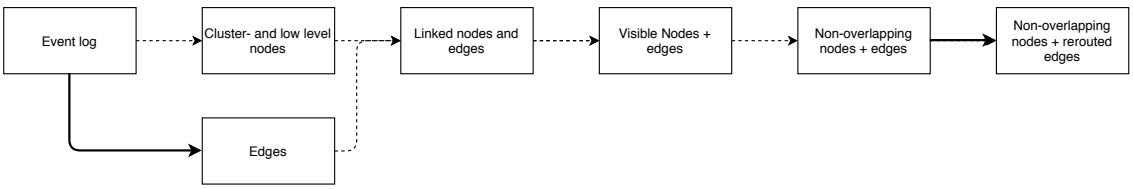


Figure 12: Figure of the different steps needed to compute the drawn graph based on an event log. The solid arrows are steps described in this chapter. The dashed arrows are steps described in Chapter 3

4.1 Process mining

In order to apply process mining, we need a case id, timestamp, and activity attribute. For the activity attribute, we need a unique activity per location. We create this attribute by adding the unique identifier of the location where that event happens to the name of the activity. This is done for every event in the log. If no such identifier is available, then the coordinates of the location can also be used. That way, the resulting names of the activities are unique for every position. Table 4.1 shows an example of what the unique activities could look like.

Algorithm
Algorithm 5 shows the algorithm that is used. After the activity per location is created a process mining algorithm is used to determine the process graph. This can be any process mining algorithm but in this example a directly-follows algorithm is used to compute the process graph. This algorithm is already implemented in the UiPath Process mining platform. The corresponding directly-follows graph contains edges between every pair of nodes which are executed directly after each other in at least one case of the process. Normally, the last step in process mining is to run a layout algorithm over the created process graph to generate the layout in which the process graph needs to be drawn. However, in this case, that is not needed as the position of the nodes is completely based on the geographical location of the activities.

After the process graph is created, the nodes are mapped to the geographical position of the location corresponding to that node. This is done by matching the geographical location of the node in the process graph to the location of the node in the hierarchy. After that, all the nodes with activities that happen at the same location are merged into a single node. These nodes that are merged in this step are the different activities that happen at the same location. The edges are then moved to the node in which the original start/end point of that edge was merged to. The resulting nodes are linked to the leaf nodes described in Section 3.2. After that the nodes are clustered together as described in Chapter 3. After that the resulting are rerouted as described below.

Algorithm 5 createLowLevelEdges(lowLevelNodes, eventLog)

```
processGraph = createProcessGraph(eventLog)
for all e ∈ processGraph.edges do
    find lowLevelNodes which are linked to start and end node of e
    Create edge between these nodes
    add created edge to list of edges
end for
return list of created edges
```

Running time
Finding the leaf nodes to which the edges link can be done in $O(\log n)$ time where n is the number of leaf nodes. All the other operations that need to be done per edge can be done in constant time. Therefore, the running time per created edge is $O(\log n)$. Therefore, the total running time of this part is $O(e * \log n)$.

Activity	Location ID	Location specific activity
"Receive invoice"	1000	"Receive invoice_1000"
"Pay invoice"	1000	"Pay invoice_1000"
"Pay invoice"	1001	"Pay invoice_1001"

Table 4.1: An example of what the activity for an activity per location could look like in (part of) an event log. Only the relevant attributes, for creating the unique activity, of an event are shown in this part of the event log.

4.2 Edge routing

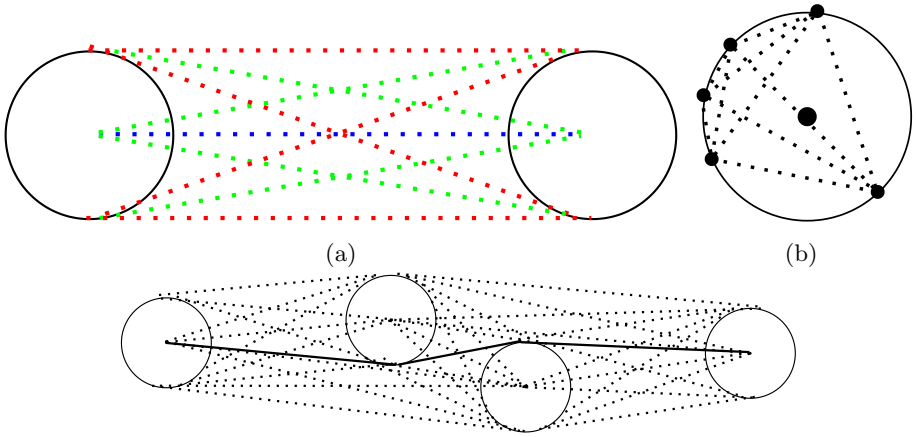
After the nodes are clustered in such a way that only non-overlapping nodes are visible, the edges need to be drawn in such a way that they do not overlap with the nodes. During the clustering step, the edges are also clustered such that there is at most one edge between a pair of nodes. The resulting edges are rerouted in this step.

Inside the nodes, information about that node are displayed which will be unreadable if there is overlap between the nodes and the edges. To prevent this, we chose to use an edge routing algorithm that completely removes the overlap between nodes and edges. This could also be fixed by putting the nodes on top of the edges so overlap between edges and nodes will not decrease the information visible on the nodes. However, with big nodes, it is hard to follow edges that go underneath a lot of different nodes. Therefore, we decided to use an algorithm which removes all the overlap between nodes and edges.

Changing the position of the displayed points could make it harder to see to which region the displayed node belongs. To make sure that the points are close to the corresponding region, we chose to use an algorithm that does not change the position of the points in order to remove the overlap between points and edges.

Algorithm
In this algorithm the edges are rerouted such that there is no overlap between nodes and edges. Figure 14b shows an example of how this would look. The way the edges are routed is based on a robot motion planning algorithm [28]. In that paper the edges are found using Dijkstra's algorithm [7] on a tangent visibility graph of the visible points. In a tangent visibility graph, besides a direct edge between every pair of nodes in the graph that is directly visible, there is also an edge for every of the four possible tangent edges between the borders of both circles. These are represented by the red edges in Figure 13a. In addition, the edges between the center of the first point and border of the other point are added. These are represented by the green edges in Figure 13a For all those edges it is separately checked whether they overlap with another node. Only the edges that do not overlap with another node are added to the visibility graph. Figure 13 shows a tangent visibility graph. In order to make it a connected graph on which Dijkstra's algorithm can be used, edges between every pair of nodes on the border of the same point are added, excluding the node in the center of the point. These edges are referred to as the point edges for the remainder of this chapter. Figure 13b shows an example of those point edges.

There is a predefined margin between the nodes and the border of the point. This margin is partially used to displace the edges going along that point a bit in order to split edges going along the same point (see Section 4.3). This margin is also there to make sure that the edges do not pass a point close to the border as this decreases the readability. If, for a certain point edge pair, the edge does not overlap with the point but the minimal distance to the point is less than the predefined margin, then this edge is handled as overlapping with that point. This is again to make sure that edges have a certain distance to all points, but also to make sure that the edge will not overlap with the point if the edge is displaced a bit as described below.

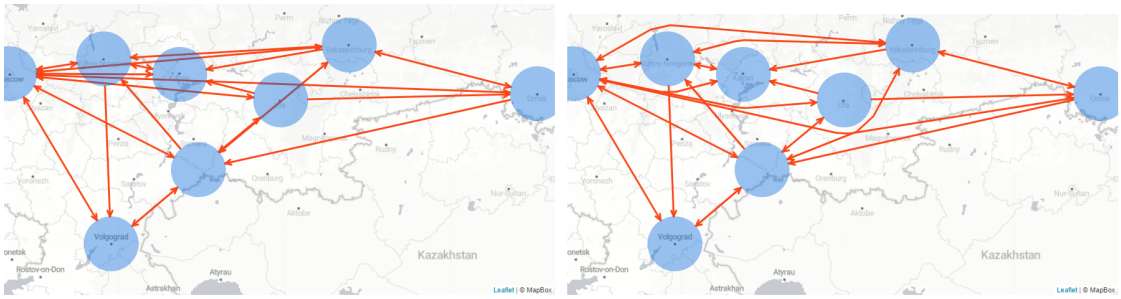


(a) (b)

(c)

Figure 13: a) The edges between two points that are added to the visibility graph. The red edges represent edges that are between the borders of both nodes. The green edges represent the edges that are between the center of and border of the nodes. The blue edge represent the edge between the center of both nodes. b) Example of the edges that are added for nodes on the border of the same point. The dashed lines are the edges added to the visibility graph. c) Example of the created edge through the edges in the visibility graph. The solid lines are the line segments of the created lines between the two points. The dashed lines are all the other edges in the visibility graph.

After the visibility graph is created, Dijkstra's algorithm is used to determine the shortest path through the visibility graph for all the edges. If we used this algorithm without changes, then it could happen that an edge would go through the center of points which are not the start- or endpoint of that point. This will create node-edge overlap, which we do not want. Therefore, the nodes in the visibility graph that are in the center of a point can only be used as start- or endpoint of a path to prevent the edges to go through the center of those nodes. Figure 14 shows a comparison between the graph before the edges are rerouted and after. There are multiple edge crossings close to the node which make it hard to follow edges through. Section 4.3 describes how these crossings are removed.



(a) (b)

Figure 14: Comparison between the edges drawn without removing node-edge overlap (a) and with removing node-edge overlap (b).

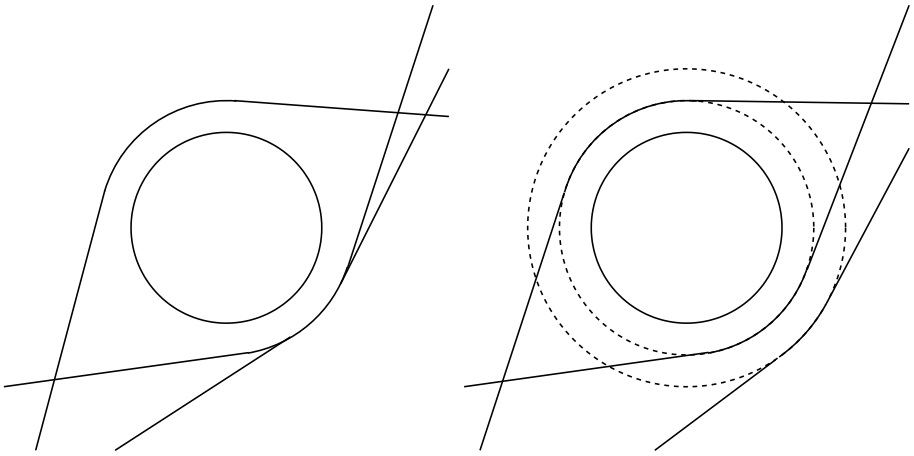
Running time
Creating the edges between two points is consists of two different kind of edges with a different running time calculation. For the edges between two nodes the number of edges created is constant. However, for every edge it needs to be checked if it is intersecting with one of the visible points. This is done using a naive approach which just checks for all points if the edge intersects with that point. Which takes $O(v)$ where v is the number of visible points. Therefore, this has a running time of $O(v)$ for every pair of visible points or $O(v^3)$ for the whole graph. For the all point edges of the same point, it does not need to be checked if they are intersecting with a point. However, The number of edges depend on the number of edges that have an endpoint on that point. In the worst-case there is at least one edge coming from all the other visible points. Therefore, in the worst case, this part of the algorithm takes $O(v)$ time for every new endpoint added to a visible point. Therefore, in the worst case, the running time to create all point edges per visible point is $O(v^2)$. This results a running time for this part of $O(v^3)$ for the whole graph. Therefore, both parts combined will give a worst-case running time of $O(v^3)$.

The edge routing itself is done using Dijkstra's algorithm. Without using Fibonacci heaps this is done in $O(e \log n)$ time. Where n is the number of nodes in the graph on which Dijkstra's algorithm is used and e is the number of edges in the graph. Note that n is not equal to the number of visible points because at every visible point there is a separate node in the graph for every edge that connects to that point. Every visible point has at least one node in the visibility graph for every other visible point which has an unobstructed edge between them. In the worst case that means $O(v)$ nodes per visible point. The nodes on the border of the same point form a fully connected sub graph. Therefore, in the worst case, the number of edges per visible point is $O(v^2)$. This results in a, worst case, total number of edges in the visibility graph of $O(v^3)$. This results in a total running time per edge of $O(v^3 \log v)$ This is done for every edge in the graph, which is in the worst-case, which is a fully connected graph, v^2 edges. So the total running time to re-route all the edges is $O(v^2 * v^3 \log v) = O(v^5 \log v)$.

4.3 Lower edge overlap

Because the same visibility graph is used for every edge it can happen that multiple edges go through the same node, or nodes very close to each other, in the visibility graph, which makes it hard to distinguish which edge is which (see Figure 15a). In order to fix this, the edges which goes through the same nodes or nodes very close to each other node need to be replaced such that they all have a unique position.

Algorithm
Algorithm 6 shows the algorithm for displacing the edge to reduce edge overlap. The general idea of this algorithm is to have several layers around the point on which edges can be placed. When an edge is put on a certain layer, the area between the start- and end node along that point is reserved for that edge and no edge can be placed in between those nodes on the same layer. The decision of which edge should be on which layer has influence on the number of edge crossings that are around that point. There are, in the worst case, $e!$ different edge placements possible. This happens when all edges need to be placed on the same area around the circle. Trying all these is not possible as that would result in an algorithm with a running time of $O(e!)$ where e is the number of edges. An alternative is to use a greedy algorithm which places the edges on the layer in an order that has a low number of edge crossings. The edges on the lowest layer can, in the worst case, intersect with an edge on every layer above for both end points. Therefore, placing the largest edges on the lowest possible layer will make sure that there are, in most cases, fewer crossings with the layers above as there are fewer edges on the lowest layer. Therefore, the edges are added to the layer in the order of the biggest angle between the start and end node along

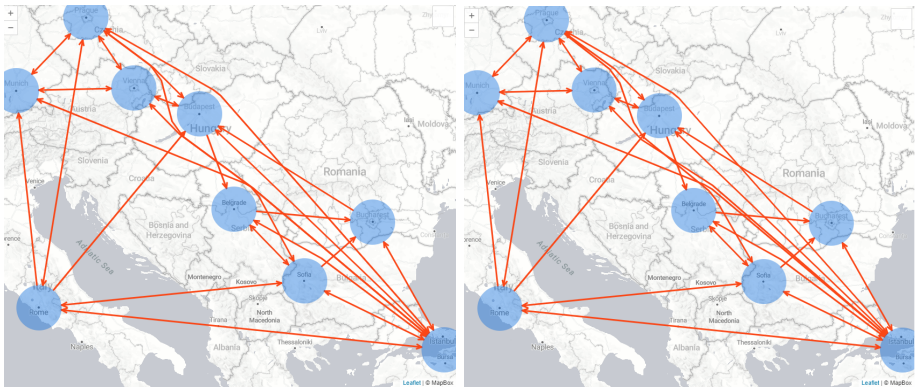


(a) (b)

Figure 15: Comparison between the edges drawn without edge displacement (a) and with edge displacement (b) around a circle.

Algorithm 6 displaceEdges(point, edges)

```
sort edges on the angle along the point between start and end node
initialize list of layers around point with entries for all nodes on point
for All edges do
    for All layers around point do
        if No other edge placed between start and endpoint of this edge then
            Place edge on this layer
            fill array for all nodes which are between start and end node of this edge
        end if
    end for
    if No Circle found then
        Add new layer to list
        Place edge on this layer
        fill array for all nodes which are between start and end node of this edge
    end if
end for
maxLayers = number of layers in the list of layers.
for All edges do
    distanceToPoint = lowestOffset + (highestOffset − lowestOffset) ∗ (layer)/maxLayer)
    for All nodes of edge along point do
        Change distance to point to distanceToPoint
    end for
end for
```



(a) (b)

Figure 16: Comparison of the overall graph between the edges drawn without edge displacement (a) and with edge displacement (b) around circles.

that point. Figure 16 shows a comparison between the edge drawing with and without the edge displacement.

To determine if an edge can be placed on a layer, we use a list, for every layer, with entries for all nodes with incoming edges on that point. The nodes in this list are in a clockwise order around the point. When we want to add an edge to a layer, we check, for every entry in the appropriate list between the two nodes on the point, if that entry is already occupied by another edge. If that is not the case for any of the entries, then the edge is added to that layer. In addition, the entries between the two nodes on the circle are set to be occupied by this edge.

After all edges are placed on a layer, the total number of layers is known. This number is used to determine the final distance between the edge and the point as that distance depends on the total number of layers and the layer in which that edge is. The formula that is used to determine the distance to the point for a particular edge is shown in Algorithm 6. lowestOffset and highestOffset in this formula depend on the total number of layers and the maximal offset possible to not overlap with the point. The formula used for those two variables are shown below.

$$lowestOffset, highestOffset = \pm min(maxOffset, edgeDistance * maxLayer/2)$$

The edgeDistance is a predefined number which is the desired distance between two edges passing along the same node. However, when there a lot of layers needed to place all edges it can happen that the edges closest to the point are placed on top of the point, which is not allowed in this visualization. In that case the distance between the edges is lowered such that all edges fit outside the point.

Running time

For a particular point all edges that passes that point needs to be displaced. In the worst-case all edges in the graph pass along that point, which are in the worst-case v^2 edges, where v is then number of visible points, in a fully connected graph. First all edges need to be sorted on the angle they make along the point. This can be done in $O(v^2 \log v)$ time. After that, all the edges need to be added to the appropriate layer. For this layers need to be evaluated to decide if the edge is placed on that order. The approach used for this takes $O(n)$ time where n is the number of nodes in the visibility graph on that point. This is in the worst case equal to the number of visible points. In the worst case a new layer needs to be added for every edge. Therefore, there could be $O(v^2)$ layers. Combined with a running time of $O(v)$ per layer, this makes the running time of

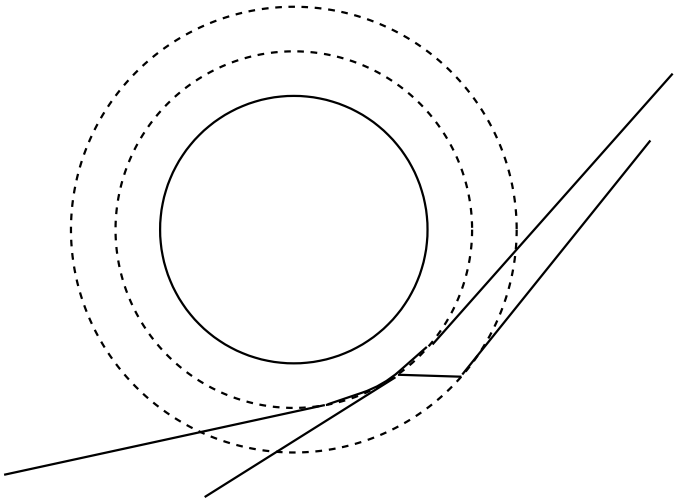


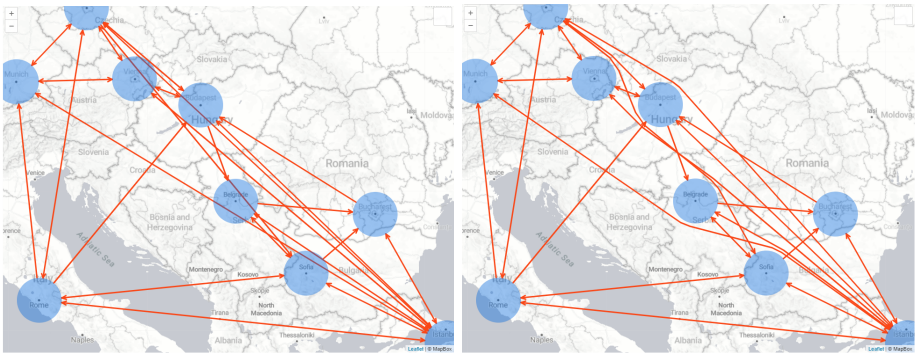
Figure 17: Example of a case where the approach based on the paper by Dokulil et al. [8] dos not give a good result. The change in rank along the circle in combination with the close distance between the points along the circle result in an edge with a sudden change in direction.

this part $O(v^3)$ per visible point. Therefore, the total running time is $O(v^2 \log(v^2) + v^3) = O(v^3)$ per visible point and $O(v^4)$ for the whole graph.

Alternative approach
Another way to fix this is based on the paper from Dokulil et al. [8]. This paper describes how to route edges through a graph of rectangle nodes. After all the edges are rerouted such that they do not overlap with the rectangle nodes, it can happen that edges go through the same point next to the corner of a rectangle. This is fixed by ranking the edges on the angle of the incoming edge and outgoing edge. The edge with the lowest combined rank of the angles of the incoming and outgoing edges placed closest to the rectangle. This can also be used for our edge routing problem because the possible locations at which the edges bend around the point are discrete. The problem is that it can happen that edges go through two nodes on the same point which are very close to each other. When the edge, in that case, has a different ranking in the second node than in the first node then there is a sudden change in distance to the circle (see Figure 17). This does not give a continuous edge. The sudden change in distance to the circle can lead to more edge crossings, which will reduce the readability of the graph. Another downside of using this algorithm is that it is not possible to place edges on a different layer when they are on different nodes, on the border of same visible point, which are close to each other. Therefore, we decided to not use this approach.

4.4 Conclusion

Figure 18 shows how the edges are drawn before and after the above mentioned algorithms are used. The overall running time of this algorithm is $O(v^6 + v^4) = O(v^6)$ where v is the number of visible nodes. Note that due to the size of the nodes, the number of visible noes that are visible are in the order of tens. Therefore, although this is a high-order polynomial running time which becomes to expensive quickly, it is still fast enough for the small number of visible nodes to not have a significant influence on the responsiveness of the map.



(a) (b)

Figure 18: Comparison of the overall graph between the edges drawn without the edge routing changes (a) and with edge routing changes (b).

Chapter 5

Evaluation

The clustering and edge routing algorithms are evaluated based on a set of metrics. These metrics focus on evaluating the created graph layout. We selected the most useful metrics, in the context of our algorithms, from the papers by Meulemans et al. [19] and Purchase et al. [22]. In addition, we created a metric that evaluates a given clustering on the similarity with the hierarchy used. We divide these metrics into two categories based on whether they evaluate the node placement or edge drawing.

The placement of the nodes is compared to a standard clustering algorithm which clusters all the nodes with the highest overlap until there are no overlapping nodes anymore. The location of the cluster node is determined by calculating the point in the center of all nodes clustered in that node. For our node clustering algorithm, we want to evaluate what the impact is on the quality of the clustering. The quality of the clustering is determined by a set of metrics which focus on how well the cluster nodes represent the underlying nodes. In addition, one of the metrics also evaluates the stability of the clustering when the view of the map changes.

The edge routing algorithm is compared with an algorithm which just draws the edges in a direct line between the two endpoints without looking at node-edge crossings. This is considered as a baseline to evaluate how the other quality aspects of edge drawing change when we completely remove node-edge crossings using this algorithm. We could also evaluate the effects of the two parts of the edge drawing separately by also calculating the metrics on the graph before the edges are displaced around the circle. However, with the small change in position for the last step, it would also have a small effect on the overall result of the metric. Therefore, we decided to not calculate the metrics before the last step.

Section 5.1 describes the procedure used to evaluate the different algorithms. Section 5.2 describes the three different datasets used for the evaluation. Section 5.3 describes the results of the metrics focused on evaluating the node clustering. Section 5.4 describes the results of the metrics focused on evaluating the edge drawing algorithm. Section 5.5 summarizes the results of the evaluation.

5.1 Procedure

The evaluation is done by setting the zoom level and center of the shown map to a set of different values. For every evaluated combination of values of those variables, the graphs are generated using the different algorithms described above. These graphs are then evaluated using the metrics below and the results are compared per position. Section 5.2 describes the used datasets.

For the evaluation, we use a set of 66 views. Appendix B shows the list of used views. Note

that not all datasets have events spread out over the whole world. Therefore, it can happen that there are no visible nodes or edges in a view for one of the datasets. In that case, we do not use the view for that dataset. Appendix B also shows which view is used for which dataset. For the stability metric, the order in which the views are evaluated is important because that metric uses the results from the view evaluated before. Therefore, the views are always evaluated in the order displayed in the list.

In total, the views are evaluated three times per dataset. Once with our clustering algorithm and our edge drawing algorithm, once with our clustering algorithm and the standard edge drawing algorithm and once with the standard clustering algorithm and our edge drawing algorithm.

5.2 Data

This evaluation process is done for three different datasets. The sections below describe the different datasets used.

5.2.1 Data set I

For the evaluation, we need an event log with the standard attributes needed for process mining (activity, timestamp, and case id). In addition, the events also need to have location attributes and separate attributes for all hierarchical levels. For this, we use a dataset containing geotagged urban activity of tourists [11]. This dataset has 11.5 million events of users "checking in" at a certain location. These check-ins happened at 2 million different locations. Every location has a category [1] associated with it. These categories form a hierarchy that divide the locations into certain use cases. For example the sub-category golf course is part of recreation. For this use case, the appropriate highest level category is used as the activity attribute to make sure that the number of different categories is not too high. In total there are 9 different high-level categories. Every "check in" event also has a timestamp attribute. We need to pre-process the data to get suitable location, hierarchy and case id attributes.

This dataset with its 2 million locations is not realistic for our main use case of company offices. Therefore, we decrease the number of different locations by clustering events to the closest city with a population higher than 1 million. These cities are obtained from a dataset consisting of every city with a population higher than 15 thousand [2]. This results in a location attribute for every event which has 361 different values.

All the cities in the above mentioned dataset can be matched to a hierarchy of at most 9 levels of the different regions in which that city is. This hierarchy is obtained using the GeoNames API [2] for all the cities in that dataset.

We can see a series of events of the same user checking in at locations as a case. Every event in the dataset has an attribute that contains a unique identifier of the user of that event. This can be used as the case identifier for this dataset. However, there are users which have more than a thousand events. In addition, there are users where there are more than two months between two consecutive events. It is not realistic to count those as a single case. Therefore, we split cases when there is more than one day between consecutive events of a user. This results, however, in a lot of cases which have only one event. We filter out those cases.

After the above mentioned filtering steps are executed, the dataset consists of roughly 3.6 million events divided over 361 locations. The events are split in roughly nine hundred thousand cases.

Table 5.1: List of countries that have events in Dataset III. The second column contains the minimum population of the cities that are used for that company. The third column contains the total number of cities that are used for that country.

Country	Min population	Number of cities
Germany	300,000	22
Austria	200,000	4
France	200,000	11
The Netherlands	200,000	5
Switzerland	200,000	1
United Kingdom	400,000	11
United States	500,000	39

5.2.2 Dataset II

Every row in this dataset represents a published paper with a list of authors and affiliations of the authors associated with it. This dataset does not contain a process. However, the clustering and edge routing algorithm does also work, with some pre-processing, for datasets containing other types of relations between locations. However, in order to execute the algorithm, the same attributes are still needed. For the activity attribute we can use the author name.

We create the cases by splitting all the rows in different events per author. All the events created from the same row get the same case id associated with it.

For the location attribute, we use the affiliation of the author. Every event has an affiliation attribute of the author. This affiliation attribute contains the name of the company or university of the author. We use the geonames API [2] to get an actual location for those names. After that, geonames is used again to get the hierarchy attributes for every location.

For the timestamp, we assign a timestamp to an event such that the order in which the events happen is the same as the order of the corresponding authors in the list of authors of the paper. This will result in edges between every pair of consecutive authors in the list of authors. We chose to do it this way because it results in edges which are similar to what it would be for actual processes.

After executing the above mentioned steps, the dataset can be used as input for the clustering and edge drawing algorithm.

5.2.3 Dataset III

This dataset contains the event log of the process of paying an invoice. Therefore, it already has the standard process mining attributes. Every event has a location attribute. However, the values of this attribute are countries. This is not precise enough to give meaningful results. Therefore we selected, for every country which has events, a list of cities. For every country with events, we selected every city with population higher than a certain minimum population. Table 5.1 shows the minimum population used and the total number of cities per country. Every event gets a random city, out of the list of cities for the country in which the event occurs, assigned to it. The hierarchy attributes are added using the geonames API. With those attributes added, it is possible to use the dataset to evaluate the algorithms.

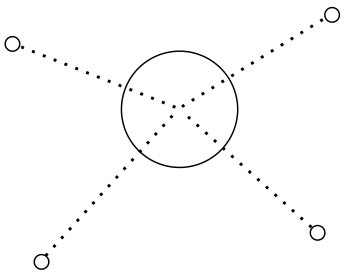


Figure 19: Example of distances calculated for the node displacement metric. The small circles are the leaf nodes. The big circle is the visible cluster node. The dashed lines are the distances that are calculated and summed for the metric.

5.3 Node clustering metrics

5.3.1 Node displacement

One of the metrics we used to evaluate the clustering algorithm is the node displacement metric. This metric is based on one of the metrics described by Meulemans et al. [19]. This metric determines, for all leaf nodes, the distance between the location of the leaf node and the location of the drawn cluster node to which that leaf node belongs. Figure 19 shows a visual example of this. Generally speaking a lower distance between the original position of the node and position of the drawn cluster node is better.

This metric is calculated by summing, for all the leaf nodes visible in the current context, the distance between the position of that leaf node on screen and the position of the visible cluster node to which that leaf node belongs on screen. The formula that is used for this metric is shown below.

$$displacement = \frac{\sum_{n \in lowLevelNodes} distance(n, n.clusterNode)}{|lowLevelNodes|}$$

When the geographical position would be used as the position in this formula the score would be higher when the shown map is zoomed out. When the view is zoomed out, the geographic distance between two overlapping nodes can be bigger. Therefore, the geographical displacement could be bigger when the nodes are merged into a node on the midpoint. Therefore, using the geographical distance to measure displacement would give higher values for zoomed out views. Therefore, we decided to use the distance on screen between the nodes. The Euclidean distance is used to calculate the distance between two nodes.

Figure 20 shows the results for the node displacement metric. It shows that overall our algorithm has a higher average node displacement than the standard clustering algorithm. Taking the average of all the measured positions gives a average 97% increase in node displacement. This increase is partially because the standard clustering algorithm will place the clustered nodes in the middle of all clustered node which has a relatively small distance to all leaf nodes in the cluster. While our algorithm places the cluster node on the position associated with that cluster node. This position is not necessarily in the middle of all the points of the cluster. Instead, this position is a predefined location for that region. Therefore, this can lead to a bigger distance between the leaf nodes and the visible cluster node.

Figure 20b and 20d show the average increase in node displacement against the number of visible points. Note that the number of visible points is not always the same for the different clustering algorithms on the same measured location. Therefore, results from the same measurements are not

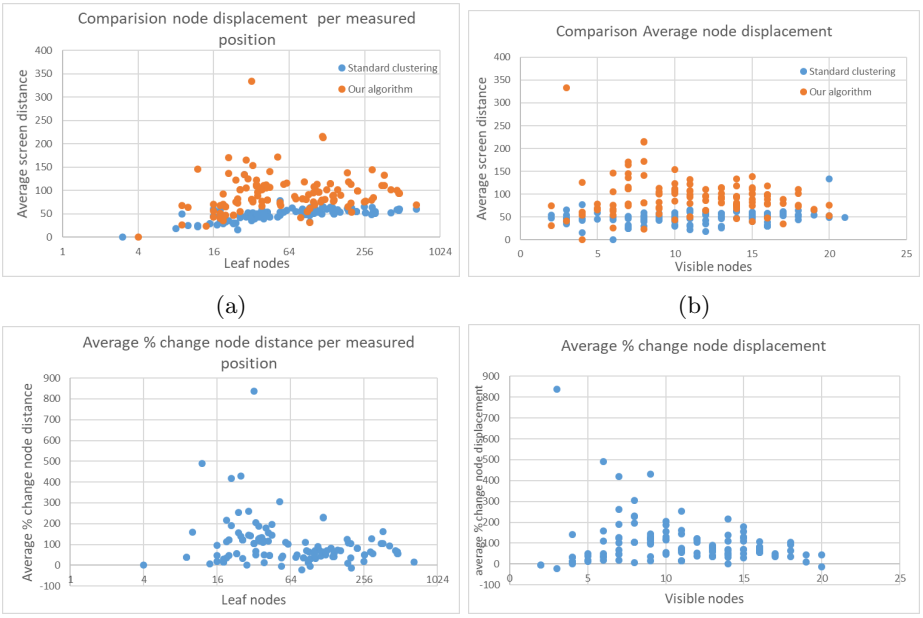


Figure 20: Results for the node displacement metric. a) and b) show the average node displacement for both algorithms. c) and d) show the % increase (or decrease) in the average node displacement. The x-axis for a and c denote the total number of leaf nodes inside the corresponding view. The x-axis for b and d denote the total number of visible nodes in the view after the clustering algorithm is used for the corresponding view.

always in the same column in this graph. The measured points which have a significantly higher node displacement than the other measured points are all cases where the number of visible points is low. One of the reasons that this could happen is that there is a cluster node with a high number of child nodes which are widespread throughout the region. When this cluster node is used the number of visible points decreases significantly and the average node displacement increases heavily because a lot of the child nodes that are clustered have a high distance to the cluster node.

Eppstein et al. [12] notes that one of the reasons why a low node displacement value, in the context of grid maps, is desired is that it helps the user in finding the location they are searching for. This is also a significant reason why the node displacement for the clustering in this context should be low. Suppose the user is searching for the cluster containing an office in The Netherlands and the only shown clusters are in Germany, France, and Denmark. Then it is not trivial to identify in which cluster the office in The Netherlands would be. Our algorithm uses the hierarchical region nodes as cluster nodes. Therefore, for those nodes it is known which region is represented by this cluster. This could be used in the visualization to make it easier to find the cluster for a particular office. In that case, a higher node displacement has a lower impact on the usability for the user.

5.3.2 Pairwise distance

This metric is based on one of the metrics described in Meulemans et al. [19]. This metric calculates, for all pairs of leaf nodes, the difference in distance between the original locations of the nodes and the used cluster nodes for both nodes. The formula used for this metric is shown below.

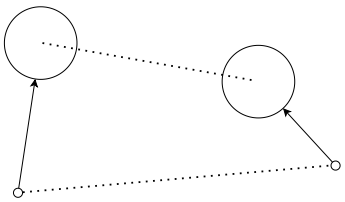


Figure 21: Example of the distances that are compared for the node-pair distance metric. The small circles are the two leaf nodes in the pair which are compared. The big circles are the visible cluster nodes. The dotted lines represent the distances used in the comparison

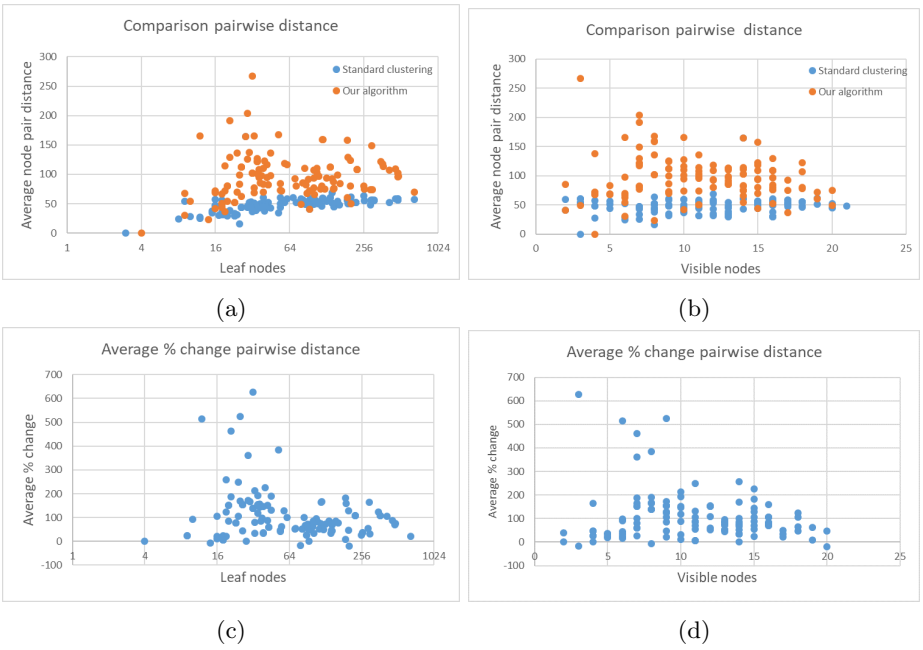


Figure 22: Results for the node pair distance metric. a) and b) show the average node pair distance. c) and d) show the % increase (or decrease) in the average node pair distance. The x-axis for a and c denote the total number of leaf nodes inside the corresponding view. The x-axis for b and d denote the total number of visible nodes in the view after the clustering algorithm is used for the corresponding view.

$$pairwiseDistance = \frac{\sum_{n,m \in lowLevelNodes} |distance(n,m) - distance(n.clusterNode, m.clusterNode)|}{|lowLevelNodes|^2}$$

When the geographical position would be used as the position in this formula the score would be higher when the shown map is zoomed out. Therefore, we use the distance on screen between the nodes instead. The Euclidean distance is used to calculate the distance between two points. Figure 21 visualizes the distances that are compared for this metric.

Figure 22 shows the results for the node pair distance metric. It shows that overall our algorithm has a higher average node pair distance than the standard clustering algorithm. Taking the average percentage change over all the measured locations gives an average increase of 105%. Note that almost all cases where our algorithm scores significantly worse then all the other cases, the number of visible nodes in our algorithm is relatively low.

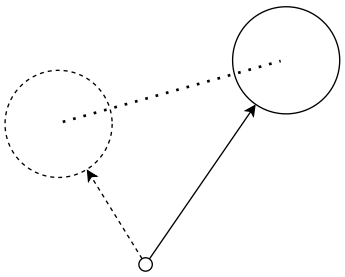


Figure 23: Visual example of how the stability metric is calculated. The small circle is the leaf node that is evaluated in this step. The dotted circle is the location of the visible cluster node that node in the old context. The other big circle is the location of the visible cluster node in the current context. The dotted line is the distance that is calculated.

Table 5.2: Table showing how often both of the algorithms are better or when they perform the equal with regards to the stability metric.

	% our algorithm better	% standard algorithm better	% equal
Zoom events	38	26	36
Drag events	26.7	38.5	34.6
Total	31.1	33.6	35.2

5.3.3 Stability

We measure the stability of the graph by evaluating the change in location of the visible cluster node of a leaf node when the view of the shown map changes. This change in view can be triggered by zooming in or out on the map or by "dragging" the map such that the center of the map is at a different location. When the location of the cluster node of a specific visible node changes less, then the stability of the graph is higher. It is easier to follow the same node throughout view changes of the map when the stability of the map is higher.

The stability, for two different views m and n of the map with leaf nodes which are in both contexts, is calculated by first checking whether the position of the visible cluster node in the first view is different than the position of the visible cluster node in the second view. This is done by comparing the geographical location because the screen position of the nodes change when the view of the map changes, the geographical location stays the same. If there is a difference between those two locations, then both positions are projected on the map, in the second view, and the distance of the positions on the screen are calculated. We use the screen position for this to better compare results between different zoom levels. The distance that is calculated is in Euclidean distance. Figure 23 shows a visual example of what the calculation looks like.

Figure 24 shows the results for the stability metric. We removed outliers with a change above 500% from the graph to increase readability. Appendix A shows the graphs with the outliers. The cases are split up into two separate graphs, Figure 24b shows the results for cases where the center of the shown view changed when comparing with the case before, also referred to as "drag events". Figure 24c shows the results for cases where the zoom level of the shown view changed, when comparing with the case before, also referred to as "zoom events". Table 5.2 shows how often both algorithms are better and the difference between drag and zoom events. This table shows that our algorithm performs better for zoom events than the standard clustering algorithm. For drag events it is the other way around. The reason that our algorithm performs better for zoom events is that when the zoom level of the view changes the area occupied by a node changes. Therefore, the nodes with which a certain node overlaps also changes. This results, in a change in which nodes are clustered. In the case of the standard clustering algorithm, the position of the

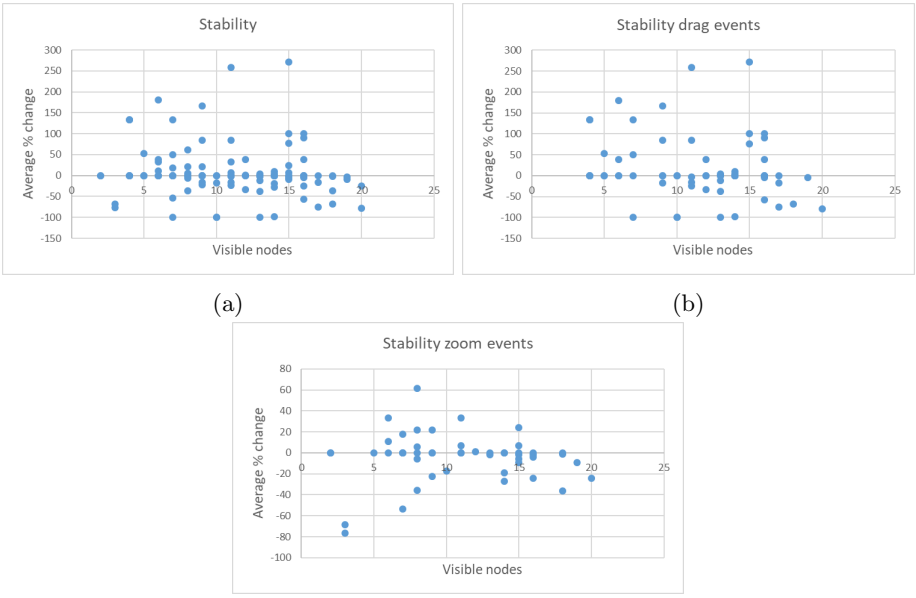


Figure 24: Results for the stability metric. a) The average percentage change in stability for drag events. b) The average percentage change in stability. c) The average percentage change in stability for zoom events. The x-axis denotes the number of visible nodes in the evaluated view.

cluster node almost always changes when the clustered nodes change. This is not the case for our algorithm due to the use of predefined locations of the cluster nodes.

5.3.4 Hierarchy

We also want to evaluate how using the hierarchy impacts which nodes are merged. For this we constructed a metric which tries to capture this. This metric will calculate, given a set of visible nodes, how well the leaf nodes clustered in the visible node corresponding to a certain hierarchy. We use this metric on the results of the standard clustering algorithm to determine the similarity between the nodes clustered using this algorithm and the hierarchy.

Algorithm 7 shows the algorithm that is used to calculate the value of this metric. Before we can use this algorithm, we need to find the lowest cluster node which has all the leaf nodes of the visible point as (indirect) child nodes. This node is then used as the "currentRootNode" parameter in the call to the algorithm. The parameter "isStartingPoint" is set to true in this call to the algorithm. The algorithm starts by checking the base case, which is if the "currentRootNode" is a leaf node in the hierarchy. If that is the case, then the algorithm will return one if the leaf node is also a leaf node for the visible node, and zero otherwise.

If the "currentRootNode" is not a leaf node, then the algorithm will make a recursive call on all the child nodes of this node and sum the resulting values. If the algorithm is calculating the value for the highest cluster node for this visible node ("isStartingPoint" = true), then we divide this summed value by the number of child nodes which has a value higher than zero, otherwise we divide by the total number of child nodes. This value is then returned. We make this distinction because there are cases where the all the leaf nodes clustered by a visible node follow the hierarchy but the hierarchy score is very low. An example of this is, when all the leaf nodes in The Netherlands and Belgium are clustered into one visible point with no other leaf nodes in it. The lowest hierarchy node in this example would be Europe. Now suppose that there are, in this example, twenty

Algorithm 7 calculateHierarchyMetric(currentRootNode, visiblePoint, isStartingPoint)

```
if currentRootNode.isLeafNode then
    if currentRootNode ∈ visiblePoint.leafNodes then
        return 1
    else
        return 0
    end if
end if
for childNode ∈ currentRootNode.childNodes do
    value += calculateHierarchyMetric(childNode, visiblePoint, false)
    summedValue += value
    if value > 0 then
        nrofChildNodesWithUsedLeaves++
    end if
end for
if isStartingPoint then
    return summedValue / nrofChildNodesWithUsedLeaves
else
    return summedValue / currentRootNode.childNodes.length
end if
```

countries in Europe which have leaf nodes in it. Then, the score without the distinction would be, in this case, $\frac{2}{20} = 0.1$. This is a very low score for a visible point, which completely follows hierarchical clustering. With the distinction in place the score is $\frac{2}{2} = 1$. Which is more suitable for this example. If this distinction would be used for all hierarchical nodes, then the algorithm will always return one as a value because by removing every node with a value of 0 on every level in the hierarchy, you remove all nodes which are not in the visible node. Therefore, only nodes are counted which are in the visible node. This results in a value of one for the metric. Therefore, we only make this distinction for the first evaluated level.

Note that by this definition of the hierarchy described above, the hierarchy score for our algorithm is always one. In our algorithm the visible points are either a cluster node or a combination of multiple cluster nodes with the same parent node. Both of those cases will give a hierarchy score of one.

Figure 26 shows the results for the hierarchy metric. In Figure 26a shows that the hierarchy score decreases as the number of leaf nodes increases. It can be assumed that leaf nodes are relatively close to other leaf nodes in the same hierarchy. Therefore, when the number of leaf nodes is low the average distance between leaf nodes is higher and the chance that leaf nodes will only merge with other leaf nodes in the same region is higher. While, when the number of leaf nodes is higher the nodes will merge with more leaf nodes and also with nodes further away, so the chance of merging with nodes outside the hierarchy. This could be one of the reasons why the hierarchy score decreases when the number of leaf nodes increases. For the relation between number of visible points and the hierarchy, it is not clear from the measured scores if there is a relation.

Overall, the standard clustering algorithm has a high hierarchy score. However, when this score is high but not one it can already have a negative impact on the readability of the graph for the user. For example, there is a graph with a node in Belgium and a node in The Netherlands, both have a high but not perfect hierarchy score. This could mean that they both have a node as leaf node which is in the other country while the user expects that the node in the Netherlands represents all nodes in The Netherlands and all nodes in Belgium for the node in Belgium respectively.

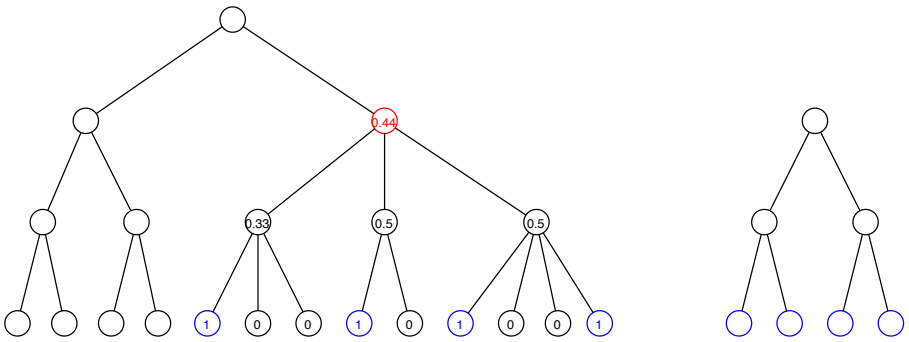
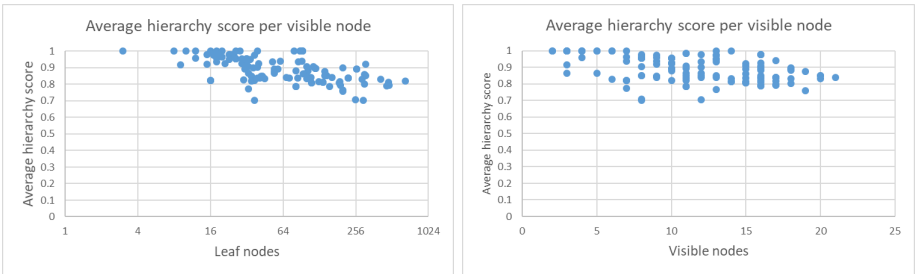


Figure 25: Example of how the value for the hierarchy metric is calculated. The tree on the right shows a visible cluster node as the root node. All the leaves are also a leaf node in the hierarchy. The tree on the left shows the hierarchy corresponding to the dataset used here. The blue leaves are the leaf nodes which are also in the visible node. The orange node is the lowest hierarchy node which has all the blue leaf nodes as (indirect) child nodes. The values inside the nodes are the metric value calculated at that node.



(a) (b)

Figure 26: Results for the hierarchy metric. a) The average hierarchy score against the number of leaf nodes in the evaluated view. b) The average hierarchy score against the number of visible nodes in the evaluated view.

5.4 Edge drawing metrics

5.4.1 Edge crossings

In this metric we calculate the average number of edge crossings per edge in the graph. Minimizing the number of edge crossings can be seen as, one of, the most important factor to increase the readability of graphs [21]. This metric is calculated by dividing the total number of edge crossings by the total number of edges in the graph. In addition, we also calculate the average smallest angle of the edge crossings. This is calculated by summing all smallest angles between edge crossings and dividing it by the total number of edge crossings.

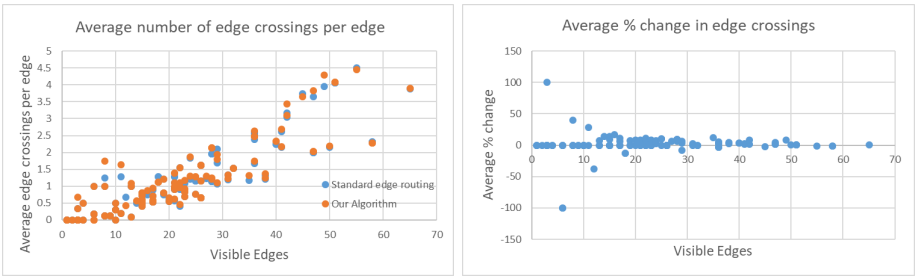
Figure 27 shows the results for the edge crossing metric. For this metric it is not possible to say that one of the algorithm performs significantly better than the other. By rerouting the edges around the circles, the same number of edges needs to be drawn on a smaller area. Which means the chance of edges crossing would increase. Therefore, we expected the number of edge crossings in our algorithm to increase. However, this is not the case.

Figure 28 shows an example of a situation in which our algorithm has a significantly better value for the edge crossing metric than the standard edge drawing algorithm. The main reason that our algorithm has a better score is that the original graph has a low number of edge crossings (in this case one). Having one less edge crossing for the same graph automatically gives a significant change in the metric score.

Figure 29 shows the results for the average edge crossing angle. It shows that the average angle does not change significantly for the different algorithms. Figure 30 shows a situation in which our algorithm performs significantly better than the standard edge drawing algorithm. The main reason for the significant difference in average edge angle is that there is only one edge crossing in this example. Therefore, the change in the angle of that edge crossing is automatically the average change for the whole graph.

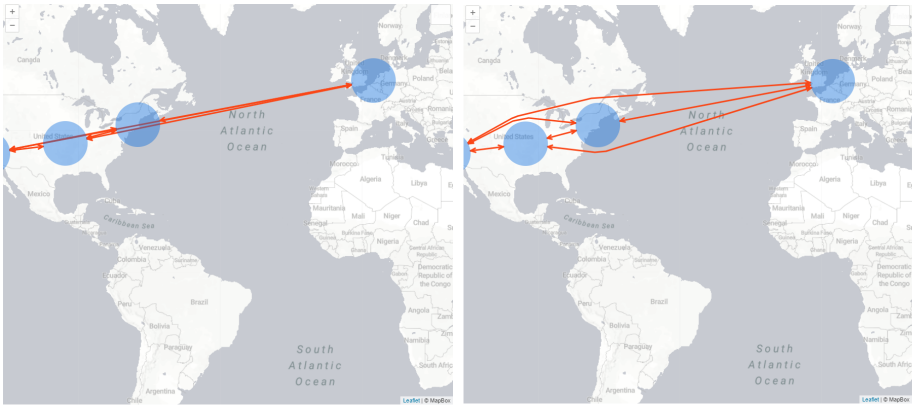
5.4.2 Angular resolution

This metric calculates to what degree the endpoints of the edges are evenly divided around the circle. When all endpoints of the edges are close it becomes harder to distinguish which edge is which [22]. Therefore, having the edges equally divided along the point increases readability. Taking the average of the angles between all the edges along the circle does not give a usable result because the average is always the same ($\frac{2\pi}{|edgesOnCircle|}$). Instead, we use the standard deviation. When the standard deviation is zero, then the edges are equally divided along the circle.



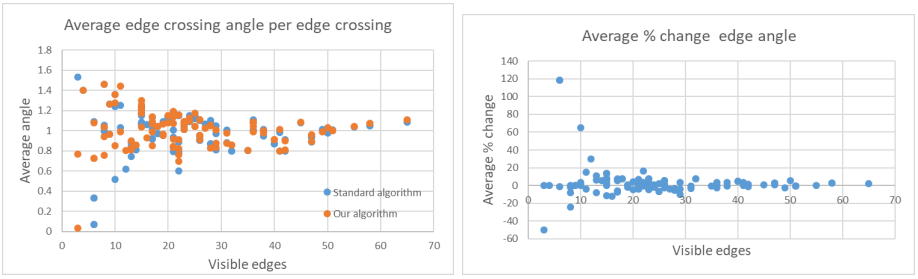
(a) (b)

Figure 27: Results for the edge crossing metric. a) The average number of edge crossings per edge for both algorithms b) The average change in number of edge crossings .



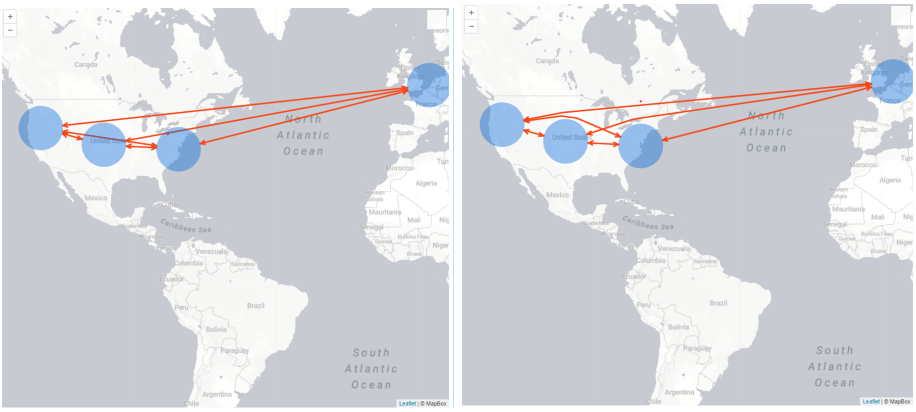
(a) (b)

Figure 28: Example of a situation in which there is a significant difference in the number of edge crossings. a) Shows the graph before edges are rerouted b) after the edges are rerouted.



(a) (b)

Figure 29: Results for the edge crossing angle metric. a) shows the average angle of the edge crossings for both algorithms b) shows the average change in angle of the edge crossings .



(a) (b)

Figure 30: Example of a situation where our algorithm performs significantly better for the edge crossing angle metric.

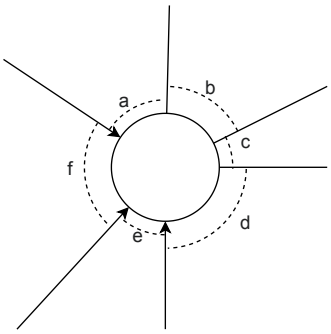
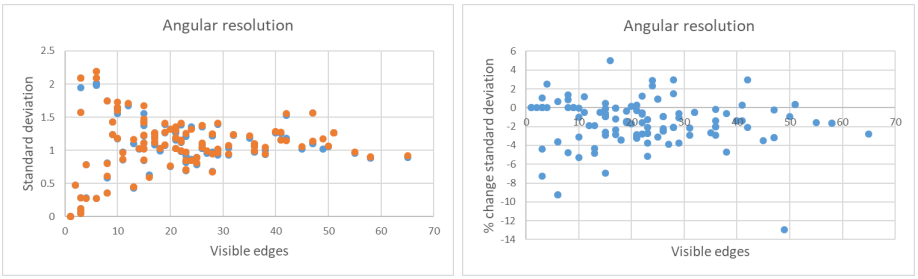


Figure 31: Example of the angles which are used in the calculation of the angular resolution.



(a) (b)

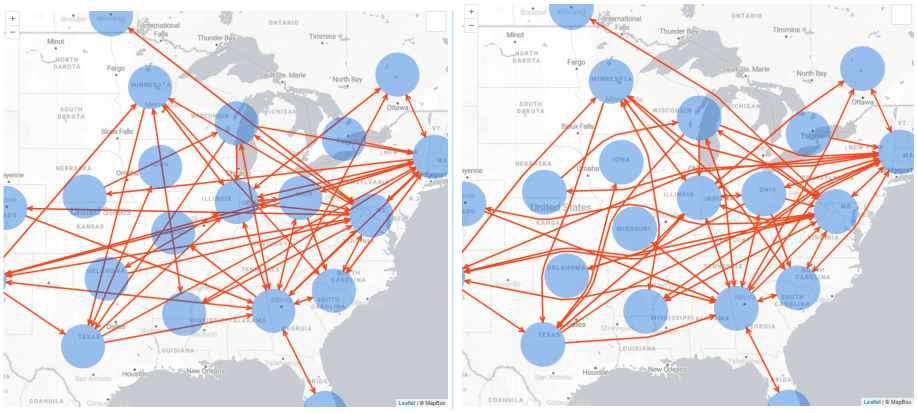
Figure 32: Results for the angular resolution metric. Both graphs show the average number of edge bends per edge. a) Shows this against the number of visible nodes in the view. b) Shows this against the number of visible edges in the view.

Figure 32 shows the results for the angular resolution metric. It shows that our algorithm performs better, occasionally, with a low number of visible edges. However, in general the angular resolution stays roughly the same for both algorithms. Therefore, our algorithm performs slightly better with regard to angular resolution then drawing the edges directly.

Figure 33 shows an example where our algorithm performs significantly better on angular resolution than the standard clustering algorithm. The main reason that our algorithm performs significantly better in this situation is that there are a lot of edges which go to or from a particular node towards different nodes which are on roughly the same line. The standard algorithm will place them on roughly the same position on the border of the node. Which results in a very low angle between those edges. In our algorithm, the edges that go to nodes on the same line will probably pass the same nodes first. The node displacement part of the algorithm, will place all those edges on a different offset along that node. Which will increase the angle between those edges on the starting node of that edge. Figure 34 visualizes what happens in this case.

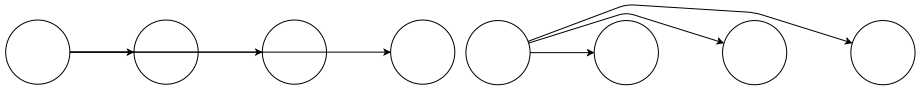
5.4.3 Edge length

In this metric we evaluate the edge length of the drawn edges. The longer the edges are, the harder it is to follow them throughout the graph. Therefore, the readability decreases when the length of the edges increase. The value of this metric is the average edge length of all edges visible in the graph. The standard edge drawing algorithm draws a direct line between the two endpoints of an edge. Therefore, the distance of this edge is the minimal distance between the two endpoints possible. This means that the value of this metric for the standard algorithm is the minimal value possible.



(a) (b)

Figure 33: Example of a situation where our algorithm performs significantly better than other situations with regard to angular resolution.



(a) (b)

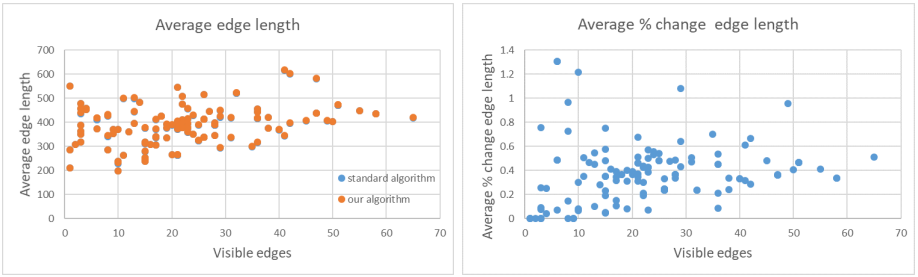
Figure 34: Example of when our algorithm positively influences the angular resolution significantly. a) The edges when the standard edge drawing algorithm is used. b) The edges when our edge drawing algorithm is used.

Figure 35 shows the results of this metric for both algorithms. The graphs show that the average edge length stays roughly the same. The highest percentage increase is 1.3%. One of the reasons why the average edge length does not increase significantly is that most graphs have a lot of edges which do not overlap with a node if they are drawn directly between the two endpoints. In those cases, the edges do not change when we use our edge drawing algorithm. This results in a lower percentage change.

Figure 36 shows the view of the map for the case where the percentage change is the highest. There are a few reasons why the metric value is relatively higher than in the other cases. The first reason is the low number of edges (only three) in combination with an edge that needs to be rerouted. With the low number of total edges the detour that one of the edges has to make to not overlap with a point has a bigger effect on the average length of the edges. The second reason is that this edge almost crosses the point in the middle of the circle. This means that the rerouted edge has to make the biggest detour possible to prevent crossing with that node. Both those reasons combined will lead to a relatively high increase in edge length. Although, with an increase of 1.3%, this is still not a significant increase in edge length.

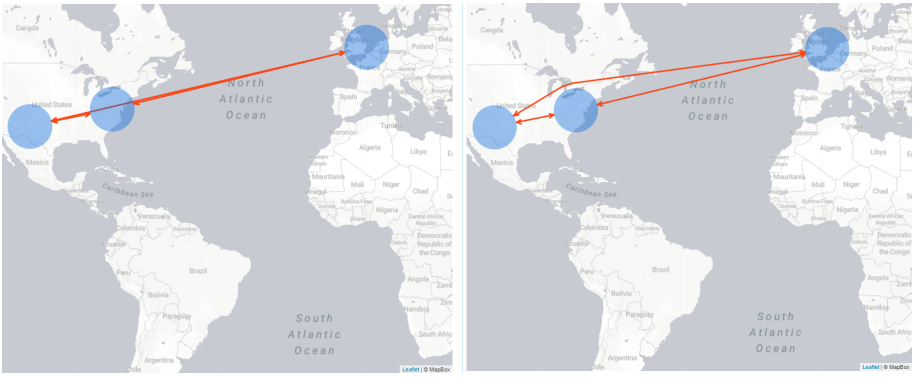
5.4.4 Edge bends

This metric calculates the average number of edge bends per edge for a given graph. Edge bends make it harder to follow the edge through the graph. Therefore, it decreases the readability of the graph [21]. The standard way of drawing edges draws an direct edge between the two points. These edges do not have any bends in it. Therefore, the value of this metric for that way of drawing edges is always zero. Every time an edge bends around a node, this is considered as one edge bend. Therefore, the number of edge bends is calculated by counting the nodes for which



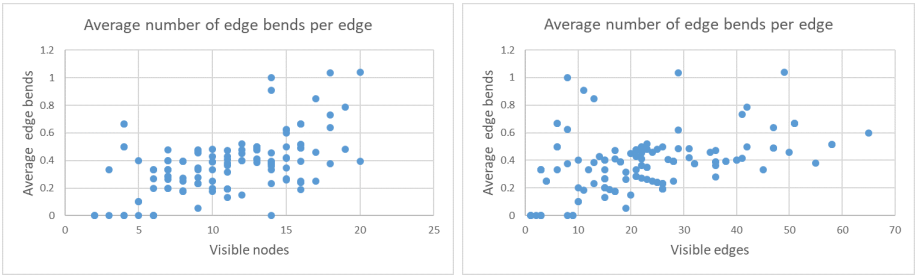
(a) (b)

Figure 35: Results for the edge length metric. a) The average edge length against the number of edge in the graph. b) The change in average edge length as a percentage by using our algorithm.



(a) (b)

Figure 36: An example of a situation with a (relatively) high percentage change in edge length.



(a) (b)

Figure 37: Results for the edge bends metric. Both graphs show the average number of edge bends per edge. a) shows this against the number of visible nodes. b) shows this against the number of visible edges

the edge bends around it.

Figure 37 shows the result for this metric. For all the measured graphs, the average number of edge bends per edge is almost always below one. Figure 37a shows that the average number of edge bends stays roughly the same when the number of nodes increases. When there are more nodes in the graph the edges have more possibilities to intersect with a node, when that happens the edge needs to be redrawn around that edge, which adds an edge bend. Therefore, we expected the average number of edge bends to depend more on the number of visible nodes.

Figure 38 shows an example of when the average number of edge bends is significantly higher than in other cases. There are several reasons why this has a significantly higher number of edge bends. The first reason is that there is a high number of visible nodes. A high number of visible nodes increases the number of possible nodes with which an edge can intersect. Therefore, increasing the number of possible nodes the edge needs to bend around. The second reason is that a lot of the drawn edges are between nodes at opposite sides of the graph. This means that the drawn edge has more nodes which it needs to be bend around to remove the overlap. Our algorithm does not take into account the number of edge bends a created edge will have. Instead, it only minimizes the length of the edge while avoiding overlap with nodes. Using an algorithm which minimizes the number of edge bends would probably not give a significantly better result in this situation. In this situation, a lot of the edge would need to be rerouted around all the nodes. Which would have a significant effect on the length of the edges. This will also decrease the readability of the graph.

5.4.5 Node-edge crossings

This metric determines the number of node-edge crossings. Node-edge crossings can make it harder to read information rendered in the nodes if the edges are on top of the nodes or can make it harder to follow edges through the graph when the edges are drawn underneath the nodes. In the latter case, this can have a significant effect, especially for graphs with large nodes, on the readability of the graph. Therefore, our algorithm completely removes node-edge crossings. However, we want to evaluate how many node-edge crossings would be in the graph if the edge would not be rerouted.

Figure 39 shows the average number of node-edge crossings per node against the total number of edges in the graph. The average number of node-edge crossings range between 0 and 2 with the actual value obviously heavily depending on the total number of edges.

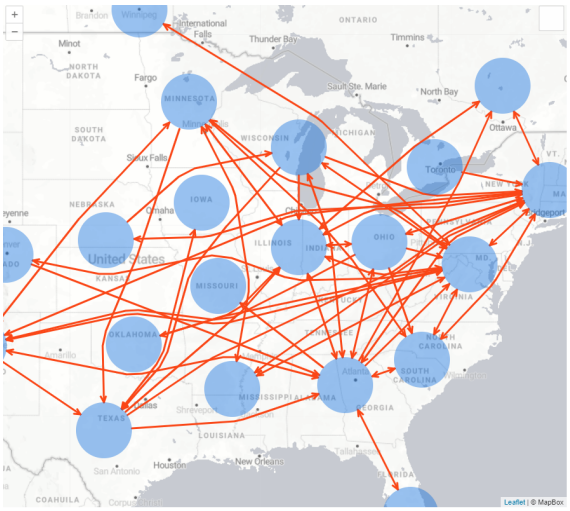


Figure 38: Example of a view with a lot of edge bends in the drawn graph.

5.5 Conclusion

From the metrics calculated for the node clustering, it can be concluded that the hierarchical clustering has a negative impact on the quality of the clustering based on the measured metrics. However, by using the hierarchy for the clustering, hierarchy specific information could be used in visualizing the cluster nodes. For example, if a bounding box of the region of a hierarchy is known, it could be used to visualize which area is represented by a cluster node. This would negate the negative impact on the clustering. During the design of the algorithm we made some decisions specifically to increase the stability of the created graph. The result of this can be (partially) seen in the results of the stability metric. For zoom event the stability is, on average, better for our algorithm. For drag events it is the other way around. From the hierarchy metric, it can be concluded that the standard clustering algorithm mostly follows the region hierarchy. However, small differences in the hierarchy and clustered nodes can already impact the readability of the graph.

It can be concluded, from the metrics calculated to evaluate the edges, that rerouting the edges in such a way that they do not overlap with the nodes, does not have a significant effect on the readability of the graph. The value of almost all metrics stay, on average, roughly the same. However, it does increase the readability of the nodes significantly when information is rendered inside the nodes.

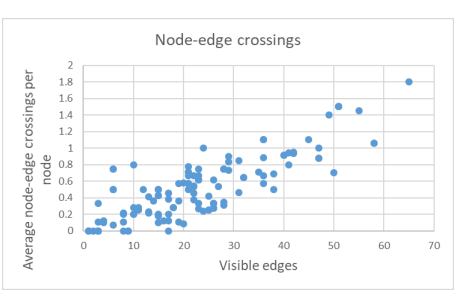


Figure 39: Average number of node-edge crossings per node

Chapter 6

Conclusions

In the introduction we identified the problem of visualizing processes in combination with geographical information. This resulted in the following research questions:

How can we visualize the geographical information of a process?

In Chapter 2 we proposed a framework to visualize a process in combination with the geographical information. This framework divides processes in four different categories based on the cardinality relation between the location and activity of events in the corresponding event log. There is a difference in the requirements of the four different categories. Therefore, we decided to propose different visualizations for the the four categories. For the many-to-many relation, there are two sub-categories based on the cardinality relation between the case and location of the events. This results in a total of five different visualizations. These visualizations either cluster the data based on the location or the activity, depending on the category to which the data belongs.

How can we cluster overlapping nodes using a region hierarchy?

In Chapter 3 we proposed an algorithm to cluster the overlapping nodes. This algorithm chooses which nodes to cluster by the position of the overlapping nodes in the hierarchy. This algorithm consists of two steps. One step which needs to be executed once to precompute all cluster nodes and a step which removes the overlap using the cluster nodes for a given view of the map.

We evaluated this algorithm in Chapter 5. We compared our algorithm against a standard clustering algorithm for overlap removal. The evaluation was done by using a set of metrics. Our algorithm performed worse on the displacement and pairwise distance metrics. For the stability metric, our algorithm performed better for zoom events and worse for drag events.

How can we draw the edges such that there are no node-edge crossings while also minimizing the effect on other graph drawing aesthetics?

In Chapter 4 we proposed an algorithm for drawing the edges based on a robot motion planning algorithm described in the paper by Wein et al. [28]. Our algorithm consists of two steps. The first steps reroutes the edges such that they do not overlap with the nodes by using the tangent visibility graph. The second step displaces the edges around the nodes to lower the number of edge crossings.

We evaluated this algorithm in Chapter 5. We compared our algorithm with a standard edge drawing algorithm that places a direct edge between the two end points. From the evaluated metrics we concluded that our edge drawing algorithm does not have a significant effect on the graph drawing aesthetics regarding edge drawing.

6.1 Future work

Use hierarchy information
With our clustering algorithm, it is known which region is represented by a given cluster node. This information could be used in the visualization to better visualize the area represented by a cluster node. This would negate the negative impact our algorithm has on the clustering algorithm. Therefore it would be useful to research how this information could be used in the visualization.

Information rendering
In this research, we focused on the steps until the graph is drawn on the map. This is, however, not the final visualization. Chapter 2 described that the visualization for the many-to-many relation would have nodes with the information rendered inside them. That is also one of the reasons why the edges are rerouted around the nodes. We did not research what the information rendered inside the nodes should be or what the impact of that is on the total visualization of the process.

In addition to the information that is always rendered on screen, the visualization described in the framework would also show extra information when the user interacts with the nodes. The proposed framework stated that this would be the process graph for that location. However, this would need more research to determine how this would be done exactly. Showing the process model for one selected location would be trivial. However, having the possibility to select multiple locations could be used to gain more insights. Therefore, more researched is needed in what is needed to visualize more locations and how that would be visualized.

Clustering
Currently the way the two nodes are chosen for which we remove the overlap is done by selecting the two nodes with the highest overlap. It would, however, be interesting to research if, and how, it would be possible to also use the hierarchy for this and what the impact would be on the clustering. For example, if the priority of choosing the nodes is on nodes with the lowest common parent node, then it could result in less clustering needed to remove all the overlap.

Bibliography

[1] foursquare categories. https://developer.foursquare.com/docs/build-with-foursquare/categories/. Accessed: 2020-07-07. 36

[2] geo names. https://www.geonames.org/. Accessed: 2020-07-07. 36, 37

[3] leaflet. https://www.https://leafletjs.com/. Accessed: 2020-07-30. 17

[4] Uipath process mining. https://www.uipath.com/product/process-mining/. Accessed: 2020-08-28. 2, 4, 17

[5] Ilya Boyandin, Enrico Bertini, Peter Bak, and Denis Lalanne. Flowstrates: An approach for visual exploration of temporal origin-destination data. In *Computer Graphics Forum*, volume 30, pages 971–980. Wiley Online Library, 2011. 11

[6] Wei Chen, Fangzhou Guo, and Fei-Yue Wang. A survey of traffic data visualization. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2970–2984, 2015. 4

[7] E Dijkstra. Dijkstra's algorithm. *Dutch scientist Dr. Edsger Dijkstra network algorithm: http://en. wikipedia. org/wiki/Dijkstra's_algorithm*, 1959. 27

[8] Jiri Dokulil and Jana Katreniakova. Edge routing with fixed node positions. In *2008 12th International Conference Information Visualisation*, pages 626–631. IEEE, 2008. 8, 32

[9] Tim Dwyer, Kim Marriott, and Peter J Stuckey. Fast node overlap removal. In *International Symposium on Graph Drawing*, pages 153–164. Springer, 2005. 8

[10] Geoffrey Ellis and Alan Dix. A taxonomy of clutter reduction for information visualisation. *IEEE transactions on visualization and computer graphics*, 13(6):1216–1223, 2007. 3

[11] emre.celikten@aalto.fi, Géraud Le Falher, and michael.mathioudakis@hiit.fi. Geotagged urban activity. 11 2017. 36

[12] David Eppstein, Marc van Kreveld, Bettina Speckmann, and Frank Staals. Improved grid map layout by point set matching. *International Journal of Computational Geometry & Applications*, 25(02):101–122, 2015. 39

[13] Diansheng Guo and Xi Zhu. Origin-destination flow data smoothing and mapping. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2043–2052, 2014. 3

[14] Robert Krueger, Qi Han, Nikolay Ivanov, Sanae Mahtal, Dennis Thom, Hanspeter Pfister, and Thomas Ertl. Bird's-eye-large-scale visual analytics of city dynamics using social location data. In *Computer Graphics Forum*, volume 38, pages 595–607. Wiley Online Library, 2019. 5, 14, 15

[15] Sander JJ Leemans, Dirk Fahland, and Wil MP van der Aalst. Discovering block-structured process models from event logs-a constructive approach. In *International conference on applications and theory of Petri nets and concurrency*, pages 311–329. Springer, 2013. 2

[16] J. W. A. Mangnus. Visualizing geographic information in event logs. Master's thesis, Eindhoven University of Technology, March 2018. 4

[17] Dennis McNamara, Jacqueline Tapia, Chihua Ma, and Timothy Luciani. Spatial analysis of employee safety using organizable event quiltmaps. In *Proceedings of the IEEE VIS 2016 Workshop on Temporal & Sequential Event Analysis*, 2016. 5

[18] Robin JP Mennens, Roeland Scheepens, and Michel A Westenberg. A stable graph layout algorithm for processes. In *Computer Graphics Forum*, volume 38, pages 725–737. Wiley Online Library, 2019. 2, 10, 11

[19] Wouter Meulemans, Jason Dykes, Aidan Slingsby, Cagatay Turkay, and Jo Wood. Small Multiples with Gaps. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):381–390, 2017. 35, 38, 39

[20] Wouter Meulemans, Nathalie Henry Riche, Bettina Speckmann, Basak Alper, and Tim Dwyer. Kelpfusion: A hybrid set visualization technique. *IEEE transactions on visualization and computer graphics*, 19(11):1846–1858, 2013. 16

[21] Helen Purchase. Which aesthetic has the greatest effect on human understanding? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1353:248–261, 1997. 45, 48

[22] Helen C. Purchase. Metrics for graph drawing aesthetics. *Journal of Visual Languages and Computing*, 13(5):501–516, 2002. 2, 35, 45

[23] David Saffo, Michail Schwab, Michelle Borkin, and Cody Dunne. Geosocialvis: Visualizing geosocial academic co-authorship networks by balancing topology-and geography-based layouts. 2020. 9

[24] Roeland Scheepens, Huub van de Wetering, and Jarke J van Wijk. Non-overlapping aggregated multivariate glyphs for moving objects. In *2014 IEEE Pacific Visualization Symposium*, pages 17–24. IEEE, 2014. 21

[25] Alice Thudt, Dominikus Baur, and Sheelagh Carpendale. Visits: A spatiotemporal visualization of location histories. In *EuroVis (Short Papers)*, 2013. 5, 12, 13

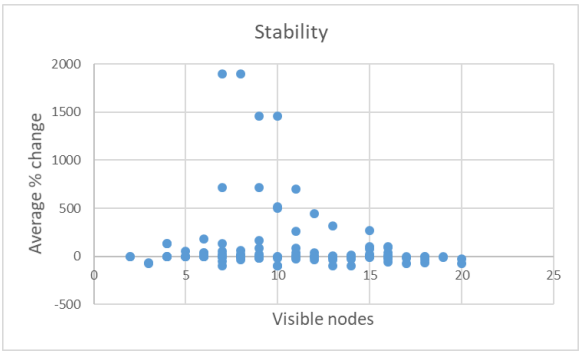
[26] Wil Van Der Aalst. Data science in action. In *Process mining*, pages 3–23. Springer, 2016. 2

[27] AJMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166:1–34, 2006. 2

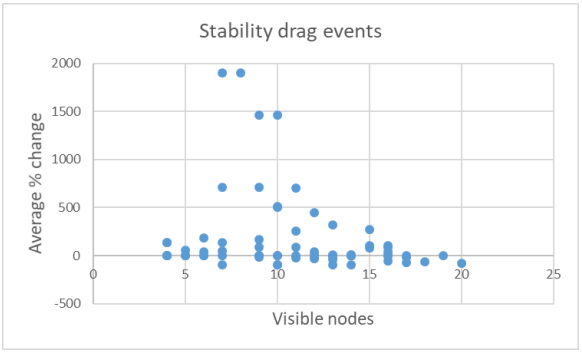
[28] Ron Wein, Jur P Van den Berg, and Dan Halperin. The visibility–voronoi complex and its applications. *Computational Geometry*, 36(1):66–87, 2007. 25, 27, 53

Appendix A

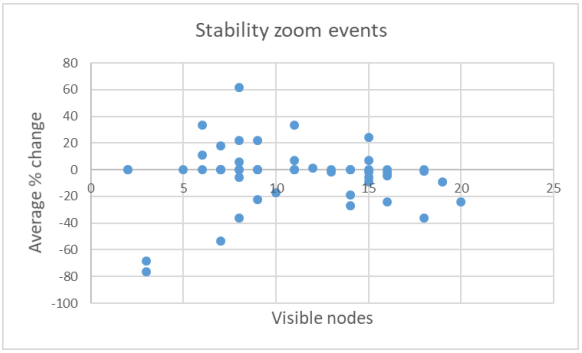
Stability Metric results



(a)



(b)



(c)

Figure 40: Results for the stability metric without outliers removed. a) shows the average percentage change in stability. b) the average percentage change in stability and c) the average percentage change in stability for zoom events

Appendix B

Used views

Latitude	Longitude	zoom	datasets
0	0	2	I, II, III
25.5	-19.5	3.1	I, II, III
25.5	-42.5	3.1	I, II, III
25.5	-61.5	3.1	I, II, III
0	-61.5	3.1	I, II, III
-22	-61.5	3.1	I, II
-22	-42.5	3.1	I, II
-22	-19.5	3.1	I, II
-22	19.5	3.1	I, II
10	19.5	3.1	I, II
10	50.5	3.1	I, II
10	80.5	3.1	I, II
47.4	15.1	4.2	I, II, III
47.4	15.1	5.4	I, II, III
47.4	8	5.4	I, II, III
47.4	15.1	5.4	I, II, III
47.4	21.1	5.4	I, II, III
40.6	15.1	5.4	I, II, III
54.4	15.1	5.4	I, II, III
49.4	15.1	6	III
54.4	15.1	6	III
49.4	15.1	6	III
49.4	19.1	6	III
54.4	19.1	6	III
54.4	0	6	III
49.4	0	6	III
45.4	0	6	III
45.4	10	6	III
49.4	10	6	III
-21.7	-52.5	, 4.3	I
-31.7	-52.5	, 4.3	I
-21.7	-52.5	, 4.3	I
-4	-52.5	, 4.3	I
-21.7	-52.5	, 4.3	I
-21.7	-52.5	, 4.9	I
-31.7	-52.5	, 4.9	I

-21.7	-52.5	, 4.9	I
-4	-52.5	, 4.9	I
40.5	-88.6	3.5	I, II, III
40.5	-88.6	4.6	I, II, III
40.5	-88.6	5.2	I, II, III
40.5	-88.6	4.6	I, II, III
40.5	-105.6	4.6	I, II, III
40.5	-88.6	4.6	I, II, III
47	-77	4.6	I, II, III
30	110.4	3.5	I, II
30	110.4	4	I, II
30	110.4	4.5	I, II
30	110.4	5	I, II
30	110.4	5.5	I, II
16	110.4	5.5	I, II
16	100.4	5.5	I, II
16	80.4	6.5	I
16	85.4	6.5	I
4	110.4	5	I, II
4	110.4	4.5	I, II
4	110.4	4	I, II
4	110.4	3.55	I, II
0	0	3	I, II
0	0	4	I, II
50	50	4	I, II
-8.4	-51.6	4	I
2.6	51.6	4	I
2.6	51.6	5	I